a2ps, version 4.9.9

a2ps General Purpose PostScript Generating Utility
Edition 4.9.9, 7 November 1997

Akim Demaille
Miguel Santana

1 Introduction

This document describes a2ps version 4.9.9. The latest versions may be found on the a2ps home page[1] .

We tried to make this document informative and pleasant. It tries to be more than a plain reference guide, and intends to offer information about the concepts or tools etc. that are related to printing PostScript.

Please, send us emailcards :). Whatever the comment is, or if you just like a2ps, write to Miguel Santana (Miguel.Santana@st.com) and Akim Demaille (demaille@inf.enst.fr).

1.1 Description

a2ps formats files for printing on a PostScript printer.

The format used is nice and compact: normally two pages on each physical page, borders surrounding pages, headers with useful information (page number, printing date, file name or supplied header), line numbering, pretty-printing, symbol substitution etc. This is very useful for making archive listings of programs.

Compared to the previous versions, a2ps version 4.9.9 provides:

1. various configuration files (see Section 4.1 [Configuration Files], page 27)
2. some powerful meta-sequences to define the headers the way you want (see Section 3.2 [Meta Sequences], page 21);
3. a totally open approach of fonts and encodings;
4. a rather strong support of Latin 2, 3, 4, 5 and 6 encodings, thanks to Ogonkify (see section "Overview" in *Ogonkify manual*), written by Juliusz Chroboczek.
5. fully customizable output style: fonts, background and foreground colors, line numbering style etc. (see Section 7.4 [Designing PostScript prologues], page 60).
6. the ability to delegate the processing of some files to other filters (see Section 4.1.9 [Your Delegations], page 31).
7. and of course, the ability to pretty-print sources written in quite a few various languages (see Chapter 6 [Pretty Printing], page 41).

1.2 Glossary

This section settles some terms that are going to be used through out this document, and provides the definitions of some terms you probably want to know about.

AFM file AFM stands for Adobe Font Metrics. These files contain everything one needs to know about a font: the width of the characters, the available characters etc.

Delegate Another filter (application) which a2ps may call to process some files. This feature is especially meant for page description files (see Section 4.1.9 [Your Delegations], page 31).

[1] http://www-inf.enst.fr/~demaille/a2ps/

Encoding
Code Set Association of human readable characters, and computers' internal numbered representation. In other words, they are the alphabets, which are different according to your country/mother tongue. E.g.: ASCII, Latin 1, corresponding to Western Europe etc.

See Section 5.1 [What is an Encoding], page 36 to know more about encodings.

Face A virtual style given to some text. For instance, *Keyword*, *Comment* are faces.

Medium Official name (by Adobe) given to the output physical support. In other words, it means the description of a sheet, e.g., A4, Letter etc.

Virtual page
Logical page
Area on a physical page in which a2ps draws the content of a file. There may be several virtual pages on a physical page.

Page A single side of a sheet.

Page Description Language
A language that describes some text (which may be enriched with pointers, pictures etc.) and its layout. HTML, PostScript, LaTeX, roff and others are such languages. A file written in those languages is not made to be read as is by a human, but to be transformed (or compiled) into a readable form.

PostScript *PostScript* is a page description language, in other words, it is a programming language, in which most programs are a list of instructions that describes lines, shades of gray, or text to draw on a page.

This is the language that most printers understand.

PostScript is a trademark of Adobe Systems Incorporated.

Sheet The physical support of the printing: it may support one or two pages, depending on your printing options.

Style sheet
Set of rules used by a2ps to give a face to the strings of a file. In a2ps, each programming language which is supported is defined via one style-sheet.

1.3 Version Numbering Scheme

a2ps is alive. To let you know its status, the version numbers are structured *à la* Linux.

What is to be understood in 'a2ps version $x.y.z.t$'?

- if y is even, then this is an official release, fully tested and checked on many platforms, and widely distributed.
- if y is odd, then
  - if there is no t, then this is a distributed beta, with brand new features, and maybe some new bugs, or lacks in the documentation. It is only available on a2ps home page[2] .

[2] http://www-inf.enst.fr/~demaille/a2ps/

– if there is *t*, then this is a true beta, only available to members of the `a2ps` mailing list, so that problems are explored and tracked down.

1.4 Reporting Bugs

We try hard to make `a2ps` portable on any Unix platform, and bug free. But sometimes there can still be bad surprises, even after having compiled and checked `a2ps` on several very different platforms.

You may encounter some of these problems yourself. In any case, please never abandon without giving us a chance. We need information from everybody so that mistakes get fixed as fast as possible.

So, if you have a problem (configuration error, compilation error, runtime error, documentation error or unclear), first check on a2ps home page[3] and in the FAQ (see Section 10.3 [FAQ], page 70) if the issue has not been addressed yet. If it is not the case, but it appears that the version of `a2ps` you have is old, consider upgrading.

If the problem persists, send us a mail (`demaille@inf.enst.fr`) in which you mention the name of your machine and OS, the version of `a2ps`, every detail you have on your compiler, and as much traces as possible (the error messages you get on the screen, or the output of `make` when it fails etc.).

Be sure to get a quick answer.

1.5 Helping the Development

If you like `a2ps` and if you feel like helping, there are several things you can do.

Translation
: The interface of `a2ps` is under `GNU gettext` which means that all the messages can be translated, without having to look at the code of `a2ps`: you don't need to be a programmer at all. All the details are available on the a2ps translation page[4] .

Style Sheets
: Since `a2ps` is evolving and getting more powerful, the style sheets should be checked and improved. There are too many so that the authors work on them. Therefore if you feel your favorite language is not honored as it should be, improve the style sheet! (see Chapter 6 [Pretty Printing], page 41 for details.)

Encodings
: `a2ps` is wide open to any 8-bit encoding. If your language is not covered today by `a2ps`, you can easily provide the support yourself. Honestly, the trickiest part is to find correct **free** fonts that support your mother tongue (see Section 5.2 [Encoding Files], page 37, to know more).

Fonts
: There are still some characters missing in Ogonkify. See the list of missing characters[5] and the Ogonkify home page[6] for details.

[3] `http://www-inf.enst.fr/~demaille/a2ps/`
[4] `http://www-inf.enst.fr/~demaille/a2ps/po/`
[5] `http://www.dcs.ed.ac.uk/home/jec/ogonkify/missing.html`
[6] `http://www.dcs.ed.ac.uk/home/jec/ogonkify/`

Documentation
: If you feel something is missing or is unclear, send us your contributions.

Porting
: Porting a program to special architectures (MS-DOS, OS/2 etc.), or building special packages (e.g., RPM) requires having an access to these architectures. If you feel like maintaining such a port, tell us.

Features
: Well, if you feel like doing something else, go ahead!

2 User's Guide

This chapter is devoted to people who don't know yet a2ps: we try to give a soft and smooth introduction to the most useful features. For a reference manual, see Chapter 3 [Invocation], page 11. For the definition of some words, see Section 1.2 [Glossary], page 1, for questions you have, see Section 10.3 [FAQ], page 70.

2.1 Purpose

a2ps is a program that takes a text file (i.e., human readable), and makes a PostScript file out of it.

2.2 How to print

To print a file 'doc.txt', just give it to a2ps: the default setting should be the one you'd like:

```
gargantua ~ $ a2ps doc.txt
[doc.txt (plain): 9 pages on 5 sheets]
[Total: 9 pages on 5 sheets] sent to the default printer
```

a2ps sent the file 'doc.txt' to the default printer, writing two columns of text on a single face of the sheet. Indeed, by default a2ps uses the option '-2', standing for two virtual pages.

2.2.1 Basics for Printing

Say you want to print the C file 'bar.c', and its header 'foo.h', on 4 virtual pages, and save it into the file 'foobar.ps'. Just hit:

```
gargantua $ a2ps foo.h bar.c -4 -o foobar.ps
[foo.h (C): 1 page on 1 sheet]
[bar.c (C): 3 pages on 1 sheets]
[Total: 4 pages on 2 sheets] saved into the file 'foobar.ps'
```

The option '-4' tells a2ps to make four virtual pages: two rows by two columns. The option '-o' (which is the short version of '--output=foobar.ps') specifies the output file. The long options must always be separated by spaces, though short options with no arguments may be grouped.

Note too that the options may be before or after the files, it does not matter.

If you send 'foobar.ps' to a printer, you'll discover that the keywords were highlighted, that the strings and comments have a different face. Indeed, a2ps is a *pretty-printer*: if it knows the (programming) language in which your file is written, it will try to make it look nice on the paper.

But too bad: 'foo.h' is only one virtual page long, and 'bar.c' takes three. Moreover, the comments are essential in those files. And even worse: the system's default printer is out of ink. Thanks god, precious options may help you:

```
gargantua $ a2ps -A4 foo.h bar.c --prologue=gray -P lw
[foo.h (C): 1 page on 1 sheet]
[bar.c (C): 3 pages on 1 sheet]
[Total: 4 pages on 1 sheet] sent to the printer 'lw'
```

Here '-A' is used to Allow several files on the same sheet, '-P lw' means to print on the printer named 'lw', and finally, the long option '--prologue' asked to use one of the alternative printing styles of a2ps. There are other available prologues (see option '--prologue' in Section 3.1.5 [Input Options], page 16), and you can even design yours (see Section 7.4 [Designing PostScript prologues], page 60).

2.2.2 Special Printers

There are three special printers defined.

The first one, void, sends the output to the trash. Its main use is to see how many pages would have been used.

```
gargantua ~ $ a2ps -P void parsessh.c
[parsessh.c (C): 33 pages on 17 sheets]
[Total: 33 pages on 17 sheets] sent to the printer 'void'
```

The second, display sends the output to ghostview, so that you can check the output without printing. Of course if you don't have ghostview, it won't work... And it is up to you to configure another displaying application (see Section 4.1.4 [Your Printers], page 29).

And the last, file saves the output into a file named after the file you printed (e.g., saves into 'foo.ps' when you print 'foo.c').

2.2.3 Using Delegations

a2ps is able to decide that it is not the right tool to do what you want, in which case it delegates the task to other programs. What you should retain from this, is, *forget that there are delegations*. Indeed, the interface with the delegations has been designed so that you don't need to be aware that they exist to use them. Do as usual.

As an example, if you need to print a PostScript file, just hit:

```
gargantua ~ $ a2ps article.ps -d
[article.ps (ps, delegated to PsNup): 7 pages on 4 sheets]
[Total: 8 pages on 4 sheets] sent to the default printer
```

Using your defaults, a2ps delegated the task to put two virtual pages per physical page to psnup, a powerful filter part of the famous psutils by Angus Duggan.

Suppose now that you want to display a Texinfo file. Then, provided you have all the programs `a2ps` needs, just hit

```
gargantua ~ $ a2ps a2ps.texi -P display
[a2ps.texi (texinfo, delegated to texi2dvi): 75 pages on 38 sheets]
[Total: 76 pages on 38 sheets] sent to the printer ‘display’
```

Once the documentation read, you know you want to print just the pages 10 to 20, and the cover. Just hit:

```
gargantua ~ $ a2ps a2ps.texi --pages=1,10-20 -d
[a2ps.texi (texinfo, delegated to texi2dvi): 13 pages on 7 sheets]
[Total: 14 pages on 7 sheets] sent to the default printer
```

A final word: compressed files can be treated in the very same way:

```
gargantua ~ $ a2ps a2ps.texi.gz -a1,10-20 -d
[a2ps.texi (compressed, delegated to Gzip-a2ps): 13 pages on 7 sheets]
[Total: 14 pages on 7 sheets] sent to the default printer
```

You should be aware that:

- the option ‘`-Z`’ enables the delegations if they are not (see ‘`--list-options`’ for your settings);
- the set of delegations is customizable, to know the delegations your `a2ps` knows, consult ‘`a2ps --list-delegations`’.

2.2.4 Checking the Defaults

If `a2ps` did not have the behavior expected, this may be because of the default settings given by your system administrator. Checking those default values is easy:


```
gargantua ~ $ a2ps --list-options
General information on a2ps:
  version            = 4.9.9
  verbosity level    = 1
  library path       =
        /inf/soft/infthes/demaille/.a2ps
        /inf/soft/infthes/demaille/share/a2ps
        /inf/soft/infthes/demaille/share/a2ps/sheets
        /inf/soft/infthes/demaille/share/a2ps/ps
Sheets:
  default paper         = A4
  layout per page       = 2 x 1 (portrait, rows first)
  borders               = yes
  compact mode          = yes
  inside margin         = 0
More stuff deleted here
```

Remember that the on-line help is always available. Moreover, if your screen is small, you may *pipe* it into `more`. Just trust this:

```
a2ps --help | more
```

2.3 Important parameters

Many things are parameterizable in `a2ps`, but two things are just essential to make sure everything goes right:

The paper Make sure that the paper `a2ps` uses is the same as your printer (See Section 3.1.2 [Sheets Options], page 13, option ‘`--medium`’).

The encoding
Make sure that the *encoding* `a2ps` uses is the same as the standard alphabet in your country (See Section 3.1.5 [Input Options], page 16, option ‘`--encoding`’).

Both values may be checked with ‘`a2ps --list-options`’.

2.4 Localizing

`a2ps` provides some Native Language Support, that is speaking your mother tongue. It uses two special features wrt non-English languages: on the one hand, the interface between you and it, and on the other hand, the format of the dates and times.

To use these features, you may set your environment variable `LANG` to the standard code representing your language:

`ca` Catalan

`cs` Czech

`de` German

`es` Spanish

`fr` French

`it` Italian

`ko` Korean

`nl` Dutch

`pl` Polish

`tr` Turkish

The problem with this approach is that a lot more than just messages and time information is affected: especially the way numbers are written changes, what may cause problems which `awk` and such.

So if you just want messages, time format and sorting order to be localized, then define:

```
set LC_MESSAGES=fr
set LC_COLLATE=fr
set LC_TIME=fr
export LC_MESSAGES
export LC_COLLATE
export LC_TIME
```

Note that the '`make install`' *must* have been done in order to have NLS support.

2.5 Interfacing with other programs

Here are some tips on how to use `a2ps` with other programs.

2.5.1 elm

Once you are in elm, hit *o* to enter in the options edition menu, hit *p* to edit the printer command, and enter '`a2ps -2gEmail --strip=3 %s -d`'. The option '`-g`' is given because there are some special escapes that `a2ps` understands, and replaces with the correct character.

2.5.2 pine

Jan Chrillesen (`jan@chrillesen.dk`) suggests us how to use `a2ps` with the Pine mailreader. Add the following to '`.pinerc`' (of course you can put it in '`pine.conf`' as well):

```
# Your printer selection
printer=a2ps -2gEmail --strip=3 -d

# Special print command if it isn't one of the standard printers
personal-print-command=a2ps -2gEmail --strip=3 -d
```

2.5.3 Netscape

This is actually valid for any program that generates PostScript that you want to postprocess with `a2ps`.

Since the data is sent through the standard input, you need to specify that it is PostScript data (see Section 4.4 [Style Sheet Files], page 34). And just to ensure the delegations are set, force '`-Z`'. Therefore, in the '`Print`' windows of Netscape, use the following command:


```
a2ps -ZEps -
```

Do not forget to tell Netscape whether your printer supports colors, and the type of paper it uses.

3 Invocation

Calling `a2ps` is fairly simple:

`a2ps` *Options... Files...*

If no *Files...* are given, `a2ps` prints its standard input. If '`-`' appears in the *Files...*, it designate the standard input too.

3.1 Command line options

To read the options and arguments that you give it, `a2ps` uses GNU `getopt`, hence:

- the options (short with arguments or long) must be separated by spaces.
- the order between options and files does not matter: '`a2ps -4m main.c`' and '`a2ps main.c -4m`' are identical.
- short options may be grouped together: '`a2ps -4mg main.c -P printer`'
- when there are no ambiguities, long options can be abbreviated, e.g., '`--list-o`' will be understood as '`--list-options`',
- '`--`' ends the options. Anything behind '`--`' is considered to be a file: '`a2ps -- -2`' prints the file '`-2`'.

Here after *bool* stands for boolean. It is considered as true (i.e. setting the option on), if *bool* is '`yes`', or '`1`'; as false if it equals '`no`' or '`0`'; and raise an error otherwise. The corresponding short option takes no arguments, but corresponds to a positive answer.

When an argument is presented by square brackets, it means that it is optional. Optional arguments must never be separated from the short option.

3.1.1 Global behavior

-q Option
--quiet Option
--silent Option

be really quiet

-v[*level*] Option
--verbose[=*level*] Option

tell what we are doing. At

- *level* = 0, report nothing,
- *level* = 1, `a2ps` just prints the total number of pages printed,
- *level* = 2 (default), it reports it for each file,
- above, it gives internal details.

-V Option
--version Option

print version and exit successfully

-=*shortcut* Option
--user-option=*shortcut* Option

use the *shortcut* defined by the user. See Section 4.1.5 [Your Shortcuts], page 29.

Note that contrary to the other options, this special construct must not be after the files to print. This is actually a bug. If somebody sees how to fix this, please send us a patch.

--debug Option

enable debugging features. They are:

- once a style sheet is read, report on the standard output its internal representation, and perform some tests on it;
- print the overall BoundingBox in PostScript;
- down load a PostScript debugger which helps understanding why a printer may reject a file.

--guess Option

`a2ps` won't print anything, but will act as `file` does: it returns for each file the style sheet it would have used to print it.

For instance, on a `C` file, you expect it to answer '`c`'.

This can be very useful on broken systems to understand why a file is printed with a bad style sheet (see Section 4.4 [Style Sheet Files], page 34).

-h Option
--help Option

print a short help, and exit successfully.

--list-options Option

Give a extensive report on `a2ps` configuration and installation.

--list-features Option

Known media, encodings, languages, prologues, printers, macro meta sequences, delegations and user options are reported. In a word, anything that you may define.

--list-delegations Option

Detailed list of the delegations. See Section 4.1.9 [Your Delegations], page 31.

--list-encodings Option

Detailed list of known encodings. See Section 5.2.3 [Some Encodings], page 38.

--list-macro-meta-sequences Option

Detailed list of the macro meta sequences. See Section 4.1.8 [Your Meta Sequences], page 30.

--list-media Option

Detailed list of known media. See Section 4.1.3 [Your Media], page 28

--list-printers Option

Detailed list of printers and named outputs. See Section 4.1.4 [Your Printers], page 29.

--list-style-sheets Option

Detailed list of the known style sheets. See Section 6.2 [Known Style Sheets], page 41.

--list-user-options Option

Detailed list of the user options. See Section 4.1.5 [Your Shortcuts], page 29

3.1.2 Sheets Options

This options specify the general layout, how the sheet should be used.

-M *medium* Option
--medium=*medium* Option

use output medium *medium*. See the output of '`a2ps --list-media`' for the list of supported media. Typical values are '`A3`', '`A4`', '`A5`', '`B4`', '`B5`', '`Letter`', '`Legal`'. '`A4dj`', '`Letterdj`' are also defined for Desk Jet owners, since that printer needs bigger margins.

-r Option
--landscape Option

print in landscape mode

-R Option
--portrait Option

print in portrait mode

--columns=*num* Option

specify the number of columns of virtual pages per physical page.

--rows=*num* Option

specify the number of rows of virtual pages per physical page.

--major=*direction* Option

specify whether the virtual pages should be first filled in rows (*direction* = '`rows`') or in columns (*direction* = '`columns`').

-1 Option

1 x 1 portrait, 80 chars/line, major rows (i.e. alias for '`--columns=1 --rows=1 --portrait --chars-per-line=80 --major=rows`').

-2 Option

1 x 1 landscape, 80 chars/line, major rows.

-3 Option

3 x 1 landscape, 80 chars/line, major rows.

-4 Option

2 x 2 portrait, 80 chars/line, major rows.

-5 Option

5 x 1 landscape, 80 chars/line, major rows.

-6 Option

3 x 2 landscape, 80 chars/line, major rows.

-7 Option

7 x 1 landscape, 80 chars/line, major rows.

-8 Option

4 x 2 landscape, 80 chars/line, major rows.

-9 Option

3 x 3 portrait, 80 chars/line, major rows.

-j Option
--borders=*bool* Option

print borders around virtual pages.

-A Option
--compact=*bool* Option

Compact mode for a sequence of files. This option allows the printing of more than one file on the same page. Otherwise, the beginning of each file will be printed in a new sheet.

--margin[=*num***]** Option

Specify the size of the margin (*num* PostScript points, or 12 points without arguments) to leave in the inside (i.e. left for the front side page, and right for the back side). This is intended to ease the binding.

3.1.3 Pages Options

This options are related to the content of the virtual pages.

--line-numbers=*number* Option

print the line numbers from *number* lines to *number* lines.

-C Option

Alias for '`--line-numbers=5`'.

-f *size* Option
--font-size=*size* Option

scale font to *size* for body text. *size* is a float number. To change the fonts used, change the current prologue (see Section 7.4 [Designing PostScript prologues], page 60.

-l *num* Option
--chars-per-line=*num* Option

Set the number of columns per page. *num* is the real number of columns devoted to the body of the text, i.e., no matter whether lines are numbered or not.

-L *num* Option
--lines-per-page=*num* Option

Set the lines per virtual page for printing. The font size is automatically scaled up to fill in the whole page. This is useful for printing preformatted documents which have a fixed number of lines per page. The minimum number of lines per page is set at 40 and maximum is at 160. If a number less than 40 is supplied, scaling will be turned off.

-m Option
--catman Option

Understand UNIX manual **output** ie: 66 lines per page and possible bolding and underlining sequences. The understanding of bolding and underlining is there by default even if '`--catman`' is not specified.

If your file is actually a UNIX manual *input*, i.e., a roff file, then depending whether you left `a2ps` delegate or not, you will get a readable version of the text described, or a pretty-printed version of the describing file (see Section 4.1.9 [Your Delegations], page 31).

-T *num* Option
--tabsize=*num* Option

set tabulator size to *num*. This option is ignored if `--interpret=no` is given.

--non-printable-format=*format* Option

specify how non-printable chars are printed. *format* can be

'`caret`' Use classical Unix representation: '`^A`', '`M-^B`' etc.

'`space`' A space is written instead of the non-printable character.

'`question-mark`'
A '`?`' is written instead of the non-printable character.

'`octal`' For instance '`\001`', '`177`' etc.

'`hexa`' For instance '`\x01`', '`\xfe`' etc.

'`emacs`' For instance '`C-h`', '`M-C-c`' etc.

3.1.4 Headers Options

These are the options through which you may define the information you want to see all around the pages.

All these options support *text* as an argument, which is composed of plain strings and meta sequences. See Section 3.2 [Meta Sequences], page 21 for details.

-B Option
--no-header Option

no page headers at all.

-b[*text*] Option
--header[=*text*] Option

set the page header

--center-title[=*text*] Option
--left-title[=*text*] Option
--right-title[=*text*] Option

Set virtual page center, left and right titles to *text*.

-u[*text*] Option
--underlay[=*text*] Option

use *text* as *under lay* (or *water mark*), i.e., in a light gray, and under every page.

--left-footer[=*text*] Option
--footer[=*text*] Option
--right-footer[=*text*] Option

Set sheet footers to *text*.

3.1.5 Input Options

-a *Page range* Option
--pages=*Page range* Option

Select the pages to print. *Page range* is a list of interval, such as '`-a1`': print only the first page, '`-a-3,4,6,10-`': print the first 3 pages, page 4 and 6, and all the page after 10 (included). Giving '`toc`' prints the table of content whatever its page number is.

The pages referred to are the *input* pages, not the output pages, that is, in '`-2`', printing with '`-a1`' will print the first virtual page, i.e., you will get half the page filled.

Note that page selection does work with the delegations (see Section 4.1.9 [Your Delegations], page 31).

-c Option
--truncate-lines=*bool* Option

Cut lines too large to be printed inside the borders. The maximum line size depends on format and font size used and whether line numbering is enabled.

-i Option
--interpret=*bool* Option

interpret tab, bs and ff chars

--end-of-line=*type* Option

Specify what sequence of characters denotes the end of line. *type* can be:

n
unix '\n'.

r
mac '\r'.

nr '\n\r'.

rn '\r\n'.

any
auto Any of the previous cases. This last case prevents the bad surprises with files from PC (trailing '^M').

-X *name* Option
--encoding=*name* Option
use input encoding *name*. See Section 5.2.3 [Some Encodings], page 38, and the result of 'a2ps --list-encodings' to know what encodings are supported. Typical values are 'ASCII', 'latin1'... 'latin6', 'iso*n*' etc.

--stdin=*filename* Option
Give the name *filename* to the files read through the standard input.

-t *name* Option
--title=*name* Option
Give the name *name* to the document. Meta Sequences can be used (see Section 3.2 [Meta Sequences], page 21).

This is used for instance in the name given to the document from within the PostScript code (so that GhostView and others can display a file with its real title, instead of just the PostScript file name).

It is **not** the name of the output. It is just a logical title.

--prologue=*prologue* Option
Use *prologue* as the PostScript prologue for a2ps. *prologue* must be in a file named '*prologue*.pro', which must be in a directory of your library path (see Chapter 4 [Library Files], page 27). Currently the available prologues are:

'bw' (black and white) the regular presentation

'gray' same presentation, but with gray shades for comments etc.

'gray2' same as gray, but light declaration (such as function calls etc.) are mapped on the same shape as the strong declarations (such as function declaration etc.).

'color' still the same idea, but with colors.

--print-anyway=*bool* Option
force binary printing. By default, the whole print job is stopped as soon as a binary file is detected. To detect such a file we make use of a very simple heuristic: if the first sheet of the file contains more than 40% of non-printing characters, it's a binary file.

-Z Option
--delegate=*bool* Option
Enable delegation of some files to delegated applications. If delegating is on, then a2ps will *not* process the file by itself, but will call an application which handles the file in another way. If delegation is off, then a2ps will process *every* file itself.

Typically most people don't want to pretty-print a PostScript source file, but want to print what describes that file. Then set the delegations on.

See Section 4.1.9 [Your Delegations], page 31 for information on delegating, and option '--list-delegations' for the applications your a2ps knows.

--toc[=*format*] Option
Generate a Table of Content, which *format* is a Meta Sequence (see Section 3.2 [Meta Sequences], page 21) processed as a PreScript file (see Chapter 8 [PreScript], page 63). If no *format* is given, don't generate the Table of Content.

Note that it is mostly useful to define a macro (see Section 4.1.8 [Your Meta Sequences], page 30), for instance, in your 'a2psrc' file:

```
MacroMetaSequence: ToC \
\\Keyword{Table of Content}\n\
#-1!f\
|$2# \\keyword{$-.20n} sheets $3s< to $3s> ($2s#) \
pages $3p<-$3p> $4l# lines\n||\
\\Keyword{End of toc}\n
```

and to give that macro name as argument to '--toc': 'a2ps *.c --toc=#(ToC)'.

Note too that you can generate only the table of content using '--pages':

```
a2ps *.c --toc=#(ToC) -atoc
```

3.1.6 Pretty Printing Options

These options are related to the pretty printing features of a2ps.

--highlight-level=*level* Option
Specify the *level* of highlighting. *level* can be

'none' no highlighting

'normal' regular highlighting

'heavy' even more highlighting.

See the documentation of the style sheets ('--list-style-sheets') for a description of 'heavy' highlighting.

-g Option
Alias for '--highlight-level=heavy'.

-E [*language*] Option
--pretty-print[=*language*] Option

With no arguments, set automatic style selection on. Otherwise, set style to *language*. Note that setting *language* to '`plain`' turns off pretty-printing. See Section 6.2 [Known Style Sheets], page 41, and the output of '`--list-style-sheets`' for the available style sheets.

--strip-level=*num* Option

Depending on the value of *num*:

0. everything is printed;
1. regular comments are not printed
2. strong comments are not printed
3. no comment is printed.

This option is valuable for instance in `java` in which case strong comments are the so called documentation comments, or in `SDL` for which some graphical editors pollutes the specification with internal data as comments.

Note that the current implementation is not satisfactory: some undesired blank lines remain. This is planed to be fixed.

3.1.7 Output Options

These are the options to specify what you want to do out of what `a2ps` produces.

-o *file* Option
--output=*file* Option

leave output to file *file*. If *file* is '`-`', leave output to the standard output.

--version-control=*type* Option

to avoid loosing a file, `a2ps` offers backup services. This is enabled when the output file already exists.

The type of backups made can be set with the `VERSION_CONTROL` environment variable, which can be overridden by this option. If `VERSION_CONTROL` is not set and this option is not given, the default backup type is 'existing'. The value of the `VERSION_CONTROL` environment variable and the argument to this option are like the GNU `Emacs` '`version-control`' variable; they also recognize synonyms that are more descriptive. The valid values are (unique abbreviations are accepted):

'`none`'
'`off`' Never make backups (override existing files).

'`t`'
'`numbered`'
Always make numbered backups.

'`nil`'
'`existing`'
Make numbered backups of files that already have them, simple backups of the others.

'`never`'
'`simple`' Always make simple backups.

--suffix=*suffix* Option

The suffix used for making simple backup files can be set with the `SIMPLE_BACKUP_SUFFIX` environment variable, which can be overridden by this option. If neither of those is given, the default is '`~`', as it is in `Emacs`.

-P *name* Option
--printer=*name* Option

send output to printer *name*. See item '`Printer:`' and '`Unknown printer:`' in Section 4.1.4 [Your Printers], page 29 and results of option '`--list-options`' to see the bindings between printer names and commands.

-d Option

send output to the default printer. See item '`DefaultPrinter:`' in Section 4.1.4 [Your Printers], page 29.

3.1.8 PostScript options

The following options are related only to variations you want to produce onto a PostScript output.

-n *num* Option
--copies=*num* Option

print *num* copies of each page

-s *num* Option
--side=*num* Option

number of sheet sides. *num* is of course 1 or 2.

-D *key*[:*value*] Option
--setpagedevice=*key*[**:***value*] Option

Pass a page device definition to the generated PostScript output. If no *value* is given, *key* is removed from the definitions.

For example, command

```
a2ps -DDuplex:true report.pre
```

prints file '`report.pre`' in duplex (two sides) mode.

Page device operators are implementation dependent but they are standardized. See Section 7.1 [Page Device Options], page 59 for details.

-S *key*[:*value*] Option
-S *key*[::*value*] Option
--statusdict=*key*[**:***value*] Option
--statusdict=*key*[**::***value*] Option

Pass a statusdict definition to the generated PostScript output. `statusdict` operators and variables are implementation dependent; see the documentation of your printer for details. See Section 7.2 [Statusdict Options], page 59 for details.

If no *value* is given, *key* is removed from the definitions.

With a single colon, pass a call to an operator, for instance:

```
a2ps -Ssetpapertray:1 quicksort.c
```

prints file '`quicksort.c`' by using paper from the paper tray 1 (assuming that printer supports paper tray selection).

With two colons, define variable *key* to equal *value*. For instance:

```
a2ps -Spapertray::1 quicksort.c
```

produces

```
/papertray 1 def
```

in the PostScript.

-k Option
--page-prefeed Option

enable page prefeeding. It consists in positioning the sheet in the printing area while the PostScript is interpreted (instead of waiting the end of the interpretation of the page before pushing the sheet). It can lead to an significant speed up of the printing.

`a2ps` quotes the access to that feature, so that non supporting printers won't fail.

-K Option
--no-page-prefeed Option

disable page prefeeding.

3.2 Meta Sequences

The meta sequences (sometimes called *escapes* in other terminologies), are some sequences of characters that will be replaced by their values. They are very much like variables.

3.2.1 Use of Meta Sequences

They are used in several places in `a2ps`:

Page markers
Headers, footers, titles and the water mark (see Section 3.1.4 [Headers Options], page 15), in general to print the name of file, page number etc. On a new sheet `a2ps` first draws the water mark, then the content of the first page, then the frame of the first page, (ditto with the others), and finally the sheet header and footers. This order must be taken into account for some meta sequences (e.g., '`$l.`', '`$l^`').

Named output
To specify the generic name of the file to produce, or how to access a printer (see Section 4.1.4 [Your Printers], page 29).

Delegation
To specify the command associated to a delegation (see Section 4.1.9 [Your Delegations], page 31).

Table of Content
To specify an index/table of content printed at the end of the job.

3.2.2 General Structure of the Meta Sequences

All format directives can also be given in format

escape width directive

where

escape In general

'`%`' meta sequences are related to general information (e.g., the current date, the user's name etc.),

'`#`' meta sequences are related to the output (e.g., the output file name) or to the options you gave (e.g., the number of virtual pages etc.), or to special constructions (e.g., enumerations of the files, or tests etc.),

'`$`' meta sequences are related to the current input file (e.g., its name, its current page number etc.),

'`\`' introduces classical escaping sequences (e.g., '`\n`', '`\f`' etc.).

width Specifies the width of the column to which the escape is printed. There are three forms for *width*

'`+`*paddinginteger*'
the result of the expansion is prefixed by the character *padding* so that the whole result is as long as *integer*. For instance '`$+.10n`' with a file name '`$n`'='`foo.c`' gives '`.....foo.c`'.

If no *padding* is given, ' ' (white space) is used.

'`-`*paddinginteger*'
Idem as above, except that completion is done on the left: '`$+.10n`' gives '`foo.c.....`'.

'*integer*' which is a short cut for '`+`*integer*'. For example, escape '`$5P`' will expand to something like '`   12`'.

directive See Section 3.2.3 [Available Meta Sequences], page 22.

3.2.3 Available Meta Sequences

Supported meta-sequences are:

'`\\`' character '`\`'

'`%%`' character '`%`'

'$$' character '$'

'##' character '#'

'#?*cond*|*if_true*|*if_false*|'
: this may be used for conditional assignment. The separator (presented here as '|') may be any character. *if_true* and *if_false* may be defined exactly the same way as regular headers, included meta sequences and the '#?' construct.

  The available tests are:

  '#?1'
  '#?2'
  '#?3' true if tag 1, 2 or 3 is not empty. See item '$t1' for explanation.

  '#?j' true if bordering is asked ('-j').

  '#?l' true if printing in landscape mode.

  '#?o' true if only one virtual page per page (i.e., '#v' is 1).

  '#?p' a page range has been specified (i.e., '#p' is not empty).

  '#?q' true if a2ps is in quiet mode.

  '#?r' true if major is rows ('--major=rows').

  '#?v' true if printing on the back side of the sheet (verso).

  '#?V' true if verbosity level is at least 2.

'#!*key*|*in*|*between*|'
: Used for enumerations. The separator (presented here as '|') may be any character. *in* and *between* are meta sequences.

  The enumerations may be:

  '#!$' enumeration of the command line options. In this case *in* in never used, but is replaced by the arguments.

  '#!f' enumeration of the input files in the other they were given.

  '#!F' enumeration of the input files in the alphabetical order of their names.

  '#!s' enumeration of the files appearing in the current sheet.

  For instance, the meta sequences 'The files printed were: #!f|$n|, |.' evaluated with input 'a2ps NEWS main.c -o foo.ps', gives 'The files printed were: NEWS, main.c.'.

  As an exception, '#!' meta sequences use the *width* as the maximum number of objects to enumerate if it is positive, e.g., '#10!f|$n|, |' lists only the ten first file names. If *width* is negative, then it does not enumerate the -*width* last objects (e.g., '#-1!f|$n|, |' lists all the files but the last).

'$(*var*)' value of the environment variable *var*

'$[*num*]' value of the *num*th argument given on the command line. Note that $[0] is the name under which a2ps has been called.

'#(*key*)' expansion of the macro meta sequence *key* (see Section 4.1.8 [Your Meta Sequences], page 30)

'#.' the extension corresponding to the current output language (e.g. 'ps').

'%*' current time in 24-hour format with seconds 'hh:mm:ss'

'$*' file modification time in 24-hour format with seconds 'hh:mm:ss'

'$#' the sequence number of the current input file

'%#' the total number of files

'%a' the localized equivalent for 'Printed by *User Name*'

'%A' the localized equivalent for 'Printed by *User Name* from *Host Name*'

'%a{*username*}'
: the localized equivalent for 'Printed by *username*'

'%A{*username*@*hostname*}'
: the localized equivalent for 'Printed by *username* from *hostname*'.

  These two are provided in the case a2ps would be used by the print service: since neither of the user name nor the host name can be known at the time the files reach a2ps, these options should be used.

'%c' trailing component of the current working directory

'%C' current time in 'hh:mm:ss' format

'$C' file modification time in 'hh:mm:ss' format

'%d' current working directory

'$d' directory part of the current file ('.' if the directory part is empty).

'%D' current date in 'yy-mm-dd' format

'$D' file modification date in 'yy-mm-dd' format

'%D{*string*}'
: format current date according to *string* with the strftime(3) function.

'$D{*string*}'
: format file's last modification date according to *string* with the strftime(3) function.

'%e' current date in localized short format (e.g., 'Jul 4, 76' in English, or '14 Juil 89' in French).

'$e' file modification date in localized short format.

'%E' current date in localized long format (e.g., 'July 4, 76' in English, or 'Samedi 14 Juillet 89' in French).

'$E' file modification date in localized long format.

'$f' full file name (with directory and suffix).

'\f' character '\f' (form feed).

'#f0'

'#f9' ten temporary file names. You can do anything you want with them, a2ps removes them at the end of the job. It is useful for the delegations (see Section 4.1.9 [Your Delegations], page 31 and for the printer commands (see Section 4.1.4 [Your Printers], page 29).

'%F' current date in 'dd.mm.yyyy' format.

'$F' file modification date in 'dd.mm.yyyy' format.

'#h' medium height in PostScript points

'$l^' top most line number of the current page

'$l.' current line number. To print the page number and the line interval in the right title, use '--right-title="$q:$l^-$l."'.

'$l#' number of lines in the current file.

'%m' the host name up to the first '.' character

'%M' the full host name

'\n' the character '\n' (new line).

'%n' the user login name

'$n' input file name without the directory part.

'%N' the user's pw_gecos field up to the first ',' character (typically his/her full name).

'$N' input file name without the directory, and without its suffix (e.g., on 'foo.c', it will produce 'foo').

'#o' name of the output, before substitution (i.e., argument of '-P', or of '-o').

'#O' name of the output, after substitution. If output goes to a file, then the name of the file. If the output is a symbolic printer (see Section 4.1.4 [Your Printers], page 29), the result of the evaluation. For instance, if the symbolic printer 'file' is defined as '> $n.%.', then '#O' returns 'foo.c.ps' when printing 'foo.c' to PostScript. '#o' would have returned 'file'.

'#p' the range of the page to print from this page. For instance if the user asked '--pages=1-10,15', and the current page is 8, then '#p' evaluates to '1-3,8'.

'$p^' number of the first page of this file appearing on the current sheet. Note that '$p.', evaluated at the end of sheet, is also the number of the last page of this file appearing on this sheet.

'$p-' interval of the page number of the current file appearing on the current sheet. It is the same as 'p^-p.', if '$p^' and '$p.' are different, otherwise it is equal to '$p.'.

'%p.' current page number

'$p.' page number for this file

'%p#' total number of pages printed

'$p#' number of pages of the current file

'$p<' number of the first page of the current file

'$p>' number of the last page of the current file

'%q' localized equivalent for 'Page %p.'

'$q' localized equivalent for 'Page $p.'

'%Q' localized equivalent for 'Page %p./%p#'

'$Q' localized equivalent for 'Page $p./$p#'

'$s<' number of the first sheet of the current file

'%s.' current sheet number

'$s.' sheet number for the current file

'$s>' number of the last sheet of the current file

'%s#' total number of sheets

'$s#' number of sheets of the current file

'%t' current time in 12-hour am/pm format

'$t' file modification time in 12-hour am/pm format

'$t1'
'$t2'
'$t3' Content of tag 1, 2 and 3. Tags are pieces of text a2ps fetches in the files, according to the style. For instance, in mail-folder style, tag 1 is the title of the mail, and tag 2 its author.

'%T' current time in 24-hour format 'hh:mm'

'$T' file modification time in 24-hour format 'hh:mm'

'#v' number of virtual sheets

'#w' medium width in PostScript points

'%W' current date in 'mm/dd/yy' format

'$W' file modification date in 'mm/dd/yy' format

3.3 Exit status

The following exit values are returned:

'0' a2ps terminated normally.

'1' an error occured.

'2' bad argument was given.

'3' unknown language was given.

4 Library Files

To be general and to allow as much customization as possible, a2ps avoids to hard code its knowledge (encodings, PostScript routines, etc.), and tries to split it in various files. Hence it needs a path, i.e., a list of directories, in which it may find the files it needs.

The exact value of this library path is available by 'a2ps --list-options'. Typically its value is:

```
gargantua ~ $ a2ps --list-options
General information on a2ps:
  version            = 4.9.9
  verbosity level    = 1
  library path       =
        /inf/soft/infthes/demaille/.a2ps
        /usr/local/share/a2ps/sheets
        /usr/local/share/a2ps/ps
        /usr/local/share/a2ps/encoding
        /usr/local/share/a2ps/afm
        /usr/local/share/a2ps
[...]
```

You may change this default path through the configuration files (see Section 4.1.1 [Your Library Path], page 27), and with the environment variable A2PS_LIBRARY.

If you plan to define yourself some files for a2ps, they should be in one of those directory.

4.1 Configuration Files

a2ps reads several files before the command line options. In the order, they are:

1. the system configuration file (usually '/etc/a2ps.cfg')
2. the user's home configuration file ('$HOME/.a2ps/a2psrc')
3. the local file ('./.a2psrc')

In these files, empty lines and lines starting with '#' are comments.

The other lines have all the following form:

Topic: Arguments

where *Topic:* is a keyword related to what you are customizing, and *Arguments* the customization. *Arguments* may be spread on several lines, provided that the last character of a line to continue is a '\'.

In the following sections, each *Topic:* is detailed.

4.1.1 Your Library Path

To define the default library path, you can use:

'LibraryPath: *path*'
: Set the library path the *path*.

'AppendLibraryPath: *path*'
: Add *path* at the end of the current library path.

'PrependLibraryPath: *path*'
: Add *path* at the beginning of the current library path.

The environment variable A2PS_LIBRARY overrides these entries.

Note that for users configuration files, it is better not to set the library path, because the system's configuration has certainly been built to cope with your system's peculiarities. Use 'AppendLibraryPath:' and 'PrependLibraryPath:'.

4.1.2 Your Default Options

'Options: *options+*'
: Give a2ps a list of command line options. *options+* is any sequence of regular command line options (see Section 3.1 [Options], page 11).

It is the correct way to define the default behavior you expect from a2ps. If for instance you want to use Letter as medium, then use:

```
Options: --medium=Letter
```

It is exactly the same as always giving a2ps the option '--medium=Letter' at run time.

The quoting mechanism is a bit painful: you need to quote the whole *options+*. For instance

```
Options: --right-title='Page $p' --center-title='Hello world'
```

is wrong. Write

```
Options: '--right-title=Page $p' '--center-title=Hello world'
```

4.1.3 Your Media

You may define the media you use. The unit is the point (1/72 of inch). There are two formats supported:

long
: in which you must give both the size of the whole sheet, and the size of the printable area:

```
# A4 for Desk Jets
#       name    w     h     llx   lly   urx   ury
Medium: A4dj    595   842   24    50    571   818
```

where *w*x*h* are the dimension of the sheet, and the four other stand for lower left x and y, upper right x and y.

short
: in which a surrounding margin of 24 points is used

```
# A4
#       name    w     h
Medium: A4      595   842
```

is the same as

```
# A4
#       name    w     h
Medium: A4      595   842   24    24    571   818
```

4.1.4 Your Printers

A general scheme is used, so that whatever the way you should address the printers on your system, the interface is still the same. Actually, the interface is so flexible, that you should understand 'named destination' when we write 'printer'.

'`Printer:` *name destination*'
'`UnknownPrinter:` *destination*'

Tell `a2ps` where to send what it produced. Both entries are related to the '`-P` *printer*' option, the first one being triggered when *printer* is equal to *name*, and the second when *name* as no `Printer:` entry.

The *destination* must be of one of the following forms:

'`|` *command*'
in which case the output is piped in *command*

'`>` *file*' in which case the output is saved into *file*.

Meta Sequences expansion is performed on *destination* (see Section 3.2 [Meta Sequences], page 21). Recall that '`#o`' is evaluated to the destination name, i.e., the argument given to '`-P`'.

For instance

```
# E.g. 'a2ps foo.c -P bar' will pipe into 'lp -d bar'
UnknownPrinter: | lp -d #o

Printer: foo > foo.ps
Printer: wc | wc
Printer: lw | lp -d printer-with-a-rather-big-name

# E.g. 'a2ps foo.c bar.h -P file' will save into 'foo.c.ps'
Printer: file > $n.#.

# E.g. 'a2ps foo.c bar.h -P home' will save into 'foo.ps'
# in user's home
Printer: home > $(HOME)/$N.#.

# Here we address a printer which is not PostScript
Printer: deskj | gs -q -sDEVICE=ljet3d -sOutputFile=- - \
         | lpr -P laserwriter -h -l
```

MS-DOS users, and non-PostScript printer owners should take advantage in getting good configuration of these entries.

'`DefaultPrinter:` *destination*'

Tell `a2ps` where to send the PostScript it produced. The *DefaultPrinter* is used when option '`-d`' is given.

```
DefaultPrinter: | lp -d dominique
```

4.1.5 Your Shortcuts

You can define some kind of 'Macro Options' which stands for a set of options.

`UserOption:` *shortcut Option*`+`

define a *shortcut*, i.e., when `a2ps` is called with '`--user-option=`*shortcut*', then consider *Option*`+`.

```
# This emulated a line printer: no features at all
# call a2ps -=lp to use it
UserOption: lp -1m --pretty-print=plain -B --borders=no
```

4.1.6 Your PostScript magic number

`a2ps` produces full DSC compliant PostScript which is something that only PS wizards may understand[1]. Adobe said

Thou shalt start you PostScript DSC compliant files with
`%!PS-Adobe-3.0`

The bad news is that some printers will reject this header. Then you may change this header without any worry since the PostScript produced by `a2ps` is also 100% PostScript level 1[2]. For instance:

`OutputFirstLine:` *magic-number*

where *magic-number* is '`%!PS-Adobe-2.0`', or just '`%!`'.

4.1.7 Your Page Labels

In the PostScript file is dropped information on where sheets begin and end, so that post processing tools know where is page 1, 2 etc. With this information can be also stored a human readable information, which is for instance what `GhostView` shows as the list of page numbers.

`a2ps` lets you define what you want in this field:

`PageLabelFormat:` *format*

where *format* is can uses Meta Sequences (see Section 3.2 [Meta Sequences], page 21). Two macro meta sequences are predefined for this: '`#(pl_short)`' and '`#(pl_long)`'.

4.1.8 Your Meta Sequences

Meta sequences are actually used so much, that it is pleasant to write sophisticated strings, that one would like not to repeat all the time. As with user short cuts, you can define macro meta sequences:

`MacroMetaSequence:` *key value*

defines '`#(`*key*`)`' to be a short cut for *value*.

As as example, here is a macro for `psnup`, which encloses all the option passing one would like. Delegations are then easier to write:

```
MacroMetaSequence: psnup psnup -#v -q #?j|-d|| #?r||-c| -w#w -h#h
```

---

[1] Basically it means the the PostScript that `a2ps` generates is **really** modular. In other words, any kind of post-processing may be applied to the files it produced.

[2] That is to say, there are no PostScript printers that don't understand these files.

4.1.9 Your Delegations

There are some files you don't really want `a2ps` to pretty-print, typically page description files (e.g., PostScript files, roff files, etc.). You can let `a2ps` delegate the treatment of these files to other applications. The behavior at run time depends upon the option '`--delegate`' (see Section 3.1.5 [Input Options], page 16).

4.1.9.1 Defining a Delegation

To define the delegation named *Delegation*, from files of type *in* to files of type *out*, by using *command*, use:

```
Delegation: name in:out command
```

command should produce the file on its standard output. Of course meta sequences substitution is performed on *command* (see Section 3.2 [Meta Sequences], page 21). In particular, *command* should use the input file '`$f`'.

```
# In general, people don't want to pretty-print PostScript files.
# Pass the PostScript files to psnup
Delegation: PsNup ps:ps \
        psselect #?V||-q| -p#?p|#p|-| $f | \
        psnup -#v -q #?j|-d|| #?r||-c| -w#w -h#h
```

Advantage should be taken from the macro meta sequences, to encapsulate the peculiarities of the various programs.

```
# Passes the options to psnup.
# The files (in and out) are to be given
MacroMetaSequence: psnup psnup -#v #?V||-q| #?j|-d|| #?r||-c| -w#w -h#h

# Passes to psselect for PS page selection
MacroMetaSequence: psselect psselect #?V||-q| -p#?p|#p|-|

# In general, people don't want to pretty-print PostScript files.
# Pass the PostScript files to psnup
Delegation: PsNup ps:ps     #(psselect) $f | #(psnup)
```

Temporary file names ('`#f0`' to '`#f9`') are available for complex commands.

```
# Pass DVI files to dvips.
# A problem with dvips is that even on failure it dumps its prologue,
# hence it looks like a success (output is produced).
# To avoid that, we use an auxiliary files and conditional call to
# psnup instead of piping.
Delegation: dvips dvi:ps    #(dvips) $f -o #f0 && #(psnup) #f0
```

4.1.9.2 Guide Line for Delegations

First of all, select carefully the applications you will use for the delegations. If a filter is known to cause problems, try to avoid it in delegations[3]. As a thumb rule, you should check that the PostScript generating applications produce files that start by:

[3] Because hiding its use into `a2ps` just makes it even more difficult to the users to know why it failed. Let them use it at hand.


```
%!PS-Adobe-3.0
```

There are some services expected from the delegations. The delegations you may write should honor:

the input file
: available through the meta sequence '`$f`'.

the medium
: the dimension of the medium selected by the user are available through '`#w`' and '`#h`'.

the page layout
: the number of virtual pages is '`#v`'.

the page range
: the page range (in a form '`1-2,4-6,10-`' for instance) is available by '`#p`'.

the verbosity level
: please, do not make your delegations verbose by default. The silent mode should always be requested, unless '`#?V`' is set.

If ever you need to have several commands, do not use '`;`' to separate them, since it may prevent detection of failure. Use '`&&`' instead.

The slogan "*the sooner, the better*" should be applied here: in the processing chain, it is better to ask a service to the first application that supports it. An example will make it clear: when processing a `DVI` file, `dvips` knows better the page numbers than `psselect` would. So a `DVI` to PostScript delegation should ask the page selection ('`#p`') to `dvips`, instead of using `psselect` later in the chain.

There is a list of possible delegations, on the page 'Delegations for a2ps'[4] . If you know others, please let us know.

4.1.9.3 Your Internal Details

There are settings that only meant for `a2ps` that you can tune by yourself.

TemporaryDirectory: *path* — Configuration Setting
: Specify a directory in which the temporary files are put. Default is '`/tmp`', or '`/var/tmp`', or '`$`*TMPDIR*' according your environment.

FileCommand: *command* — Configuration Setting
: The command to run to call `file(1)` on a file. If possible, make it follow the symbolic links.

[4] `http://www-inf.enst.fr/~demaille/a2ps/delegations.html`

4.2 Map Files

Many things are defined through files. There is a general scheme to associate an object to the files to use: map files. They are typically used to:

- resolve aliases. For instance the ISO-8859-1 encoding is also called ISO Latin 1, or Latin 1 for short. The 'encoding.map' file why map these three names to the same Encoding Description File.
- cope with broken files systems. For instance, the-one-you-know-I-don't-need-to-name cannot handle files named 'Courier-BoldOblique.afm': it is the same as 'Courier-Bold.afm'. The 'fonts.map' file is here to associate a font file name to a font name.

The syntax of these files is:

- any empty line, or any line starting by a '#' is a comment.
- a line with the format

  `***` *path*

  requests that the file designated by *path* be included at this point.
- any other line has the format

  key *value*

  meaning that when looking for *key* (e.g., name of a font, an encoding etc.), a2ps should use *value* (e.g., font file name, encoding description file name etc.).

The map files used in a2ps are:

'encoding.map'
: Resolving encodings aliases

'fonts.map'
: Mapping font names to font file names.

'sheets.map'
: Rules to decide what style sheet to use.

4.3 Font Files

Even when a PostScript printer knows the fonts you want to use, using these fonts requires some description files.

4.3.1 Fonts Map File

See Section 4.2 [Map Files], page 33, for a description of the map files. These file associate the *font-key* to a *font* name. For instance:

```
Courier                pcrr
Courier-Bold           pcrb
Courier-BoldOblique    pcrbo
Courier-Oblique        pcrro
```

associates to font named `Courier`, the key `pcrr`. To be recognized, the font name must be exact: `courier` and `COURIER` are not admitted.

A small tool is provided with a2ps: `make_fonts_map.sh`. It collects the information on the font files a2ps can see, and generates a valid font map file, 'fonts.map.new'.

4.3.2 Fonts Description Files

There are two kinds of data a2ps needs to use a font:

- the AFM file ('*font-key*.afm'), which describe the metrics information corresponding to *font*;
- in the case *font* is not known from the printer, the PFA or PFB file which are down loaded to the printer. These files are actually the PostScript programs which execution produces the characters to be drawn on the page, in this *font*.

4.3.3 Adding More Font Support

FIXME: A few words on how to make a2ps use more fonts. Don't forget some words on broken fonts such as those seen for Cyrillic support.

4.4 Style Sheet Files

The style sheets are defined in various files. See see Chapter 6 [Pretty Printing], page 41 for the structure of these files. As for most other features, there is main file, a road map, which defines in which condition a style sheet should be used (see Section 4.2 [Map Files], page 33). This file is 'sheets.map'.

It format is simple:

'*pattern style-key*'
: if the current file name matches *pattern*, then select style *style-key* (i.e. file '*style-key*.ssh').

'`**` *pattern style-key*'
: if the result of a call to `file(1)` matches *pattern*, then select style *style-key*. Note that only the first two fields of the answer of `file(1)` are considered. E.g., in the following case

```
gargantua $ file a2ps
a2ps:   ELF 32-bit MSB executable SPARC Version 1,\
        dynamically linked, stripped
```

'ELF32-bit' will be matched against *pattern*.

The special *style-key* 'binary' tells a2ps to consider that the file should not be printed, and will be ignored, unless option '--print-anyway' is given.

If a style name can't be found, the plain style is used.

Two things are to retain from this:

1. a2ps won't guess anything for a file given through the standard input.

2. if `file` is wrong on some files, `a2ps` may use bad style sheets. In this case, do try option '`--guess`', compare it with the output of `file`, and if the culprit is `file`, go and complain to your system administrator :-), or fix it by defining your own filename pattern matching rules.

5 Encodings

`a2ps` is trying to support the various usual encodings that its users use. This chapter presents what an encoding is, how the encodings support is handled within `a2ps`, and some encodings it supports.

5.1 What is an Encoding

This section is actually taken from the web pages of Alis Technologies inc.[1]

Document encoding is the most important but also the most sensitive and explosive topic in Internet internationalization. It is an essential factor since most of the information distributed over the Internet is in text format. But the history of the Internet is such that the predominant - and in some cases the only possible - encoding is the very limited ASCII, which can represent only a handful of languages, only three of which are used to any great extent: English, Indonesian and Swahili.

All the other languages, spoken by more than 90% of the world's population, must fall back on other character sets. And there is a plethora of them, created over the years to satisfy writing constraints and constantly changing technological limitations. The ISO international character set registry contains only a small fraction; IBM's character registry is over three centimeters thick; Microsoft and Apple each have a bunch of their own, as do other software manufacturers and editors.

The problem is not that there are too few but rather too many choices, at least whenever Internet standards allow them. And the surplus is a real problem; if every Arabic user made his own choice among the three dozen or so codes available for this language, there is little likelihood that his "neighbor" would do the same and that they would thus be able to understand each other. This example is rather extreme, but it does illustrate the importance of standards in the area of internationalization. For a group of users sharing the same language to be able to communicate,

1. the code used in the shared document must always be identified (labeling)
2. they must agree on a small number of codes - only one, if possible (standards);
3. their software must recognize and process all codes (versatility)

Certain character sets stand out either because of their status as an official national or international standard, or simply because of their widespread use.

First off, there is the ISO 8859 standards series that standardize a dozen character sets that are useful for a large number of languages using the Latin, Cyrillic, Arabic, Greek and Hebrew alphabets. These standards have a limited range of application (8 bits per character, a maximum of 190 characters, no combining) but where they suffice (as they do for 10 of the 20 most widely used languages), they should be used on the Internet in preference to other codes. For all other languages, national standards should preferably be chosen or, if none are available, a well-known and widely-used code should be the second choice.

[1] `http://www.alis.com/`

Even when we limit ourselves to the most widely used standards, the overabundance remains considerable, and this significantly complicates life for truly international software developers and users of several languages, especially when such languages can only be represented by a single code. It was to resolve this problem that both Unicode and the ISO 10646 International standard were created. Two standards? Oh no! Their designers soon realized the problem and were able to cooperate to the extent of making the character set *repertoires* and coding identical.

ISO 10646 (and Unicode) contain over 30,000 characters capable of representing most of the living languages within a single code. All of these characters, except for the *Han* (Chinese characters also used in Japanese and Korean), have a name. And there is still room to encode the missing languages as soon as enough of the necessary research is done. Unicode can be used to represent several languages, using different alphabets, within the same electronic document.

5.2 Encoding Files

The support of the encodings in `a2ps` is completely taken out of the code. That is to say, adding, removing or changing anything in its support for an encoding does not require programming, not even being a programmer.

See Section 5.1 [What is an Encoding], page 36, if you want to know more about this.

5.2.1 Encoding Map File

See Section 4.2 [Map Files], page 33, for a description of the map files.

The meaningful lines of the '`encoding.map`' file have the form:

```
alias      key
iso-8859-1 latin1
latin1     latin1
l1         latin1
```

where

alias specifies any name under which the encoding may be used. It influences the option '`--encoding`', but also the encodings dynamically required, as for instance in the `mail` style sheet (support for MIME).

When *encoding* is asked, the lower case version of *encoding* must be equal to *alias*.

key specifies the prefix of the file describing the encoding ('*key*`.edf`', Section 5.2.2 [Encoding Description Files], page 37).

5.2.2 Encoding Description Files

The encoding description file describing the encoding *key* is named '*key*`.edf`'. It is subject to the same rules as any other `a2ps` file:

- please make the name portable: alpha-numerical, at most 8 characters,
- empty lines and lines starting by '`#`' are ignored.

The entries are

'`Name:`' Specifies the full name of the encoding. Please, try to use the official name if there is one.

```
Name: ISO-8859-1
```

'`Documentation/EndDocumentation`'
Introduces a note on the encoding. Typical informations expected are the other important names this encoding has, and the languages it covers.

```
Documentation
Also know as ISO Latin 1, or Latin 1.  It is a superset
of ASCII, and covers most West-European languages.
EndDocumentation
```

'`Substitute:`'
Introduces a font substitution. The most common fonts (e.g., `Courier`, `Times-Roman`...) do not support many encodings (for instance it does not support Latin 2). To avoid that Latin 2 users have to replace everywhere calls to `Courier`, `a2ps` allows to specify that whenever a font is called in an encoding, then another font should be used.

For instance in '`latin2.edf`' one can read:

```
# Fonts from Ogonkify offer full support of ISO Latin 2
Substitute: Courier                Courier-Ogonki
Substitute: Courier-Bold           Courier-Bold-Ogonki
Substitute: Courier-BoldOblique    Courier-BoldOblique-Ogonki
Substitute: Courier-Oblique        Courier-Oblique-Ogonki
```

'`Default:`'
Introduces the name of the font that should be used when a font (not substituted as per the previous item) is called but provides to poor a support of the encoding. The `Courier` equivalent is the best choice.

```
Default: Courier-Ogonki
```

'`Vector:`' Introduces the PostScript encoding vector, that is a list of the 256 PostScript names of the characters. Note that only the printable characters are named in PostScript (e.g., '`bell`' in ASCII (`^G`) should not be named). The special name '`.notdef`' is to be used when the character is not printable.

Warning. Make sure to use real, official, PostScript names. Using names such as '`c123`' may be the sign you use unusual names. On the other hand PostScript names such as '`afii8879`' are common.

5.2.3 Some Encodings

Most of the following information is a courtesy of Alis Technologies inc.[2] . See Section 5.1 [What is an Encoding], page 36 presents an instructive presentation of the encodings.

The known encodings are:

[2] `http://www.alis.com/`

ASCII ('ascii.edf') Encoding

US-ASCII.

CP1250 ('cp1250.edf') Encoding

Microsoft's CP-1250 encoding (aka CeP).

HPRoman ('hp.edf') Encoding

The 8 bits Roman encoding for HP.

IBMPC ('ibmpc.edf') Encoding

ISO-8859-1 ('latin1.edf') Encoding

The ISO-8859-1 character set, often simply referred to as Latin 1, can represent most Western European languages including: Albanian, Catalan, Danish, Dutch, English, Faroese, Finnish, French, Galician, German, Irish, Icelandic, Italian, Norwegian, Portuguese, Spanish and Swedish.

ISO-8859-10 ('latin6.edf') Encoding

Latin 6 (or ISO-8859-10) adds the last letters from Greenlandic and Lapp which were missing in Latin 4, and thereby covers all Scandinavia.

Support is provided thanks to Ogonkify.

ISO-8859-2 ('latin2.edf') Encoding

The Latin 2 character set supports the Slavic languages of Central Europe which use the Latin alphabet. The ISO-8859-2 set is used for the following languages: Czech, Croat, German, Hungarian, Polish, Romanian, Slovak and Slovenian.

Support is provided thanks to Ogonkify.

ISO-8859-3 ('latin3.edf') Encoding

This character set is used for Esperanto, Galician, Maltese and Turkish.

Support is provided thanks to Ogonkify.

ISO-8859-4 ('latin4.edf') Encoding

Some letters were added to the ISO-8859-4 to support languages such as Estonian, Latvian and Lithuanian. It is an incomplete precursor of the Latin 6 set.

Support is provided thanks to Ogonkify.

ISO-8859-5 ('iso5.edf') Encoding

The ISO-8859-5 set is used for various forms of the Cyrillic alphabet. It supports Bulgarian, Byelorussian, Macedonian, Serbian and Ukrainian.

The Cyrillic alphabet was created by St. Cyril in the 9th century from the upper case letters of the Greek alphabet. The more ancient Glagolithic (from the ancient Slav glagol, which means "word"), was created for certain dialects from the lower case Greek letters. These characters are still used by Dalmatian Catholics in their liturgical books. The kings of France were sworn in at Reims using a Gospel in Glagolithic characters attributed to St. Jerome.

Note that Russians seem to prefer the KOI8-R character set to the ISO set for computer purposes. KOI8-R is composed using the lower half (the first 128 characters) of the corresponding American ASCII character set.

ISO-8859-9 ('latin5.edf') Encoding

The ISO 8859-9 set, or Latin 5, replaces the rarely used Icelandic letters from Latin 1 with Turkish letters.

Support is provided thanks to Ogonkify.

KOI8-R ('koi8r.edf') Encoding

FIXME: Give information on KOI-8R

Macintosh ('mac.edf') Encoding

For the Macintosh encoding. The support is not sufficient, and a lot of character may be missing at the end of the job.

6 Pretty Printing

The main feature of `a2ps` is its pretty-printing capabilities. Two different levels of pretty printing can be reached:

- basic (option '`-E`') in which what you print is what you wrote.
- with graphic substitution (option '`-g`'). In this mode, some keywords are replaced by a Symbol character which best represents them. For instance, in most languages '`<=`' and '`>=`' will be replaced by the corresponding single character from the font Symbol.

6.1 Syntactic limits

`a2ps` is *not* a powerful syntactic pretty-printer: it just handles lexical structures, i.e., if in your favorite language

```
IF IF == THEN THEN THEN := ELSE ELSE ELSE := IF
```

is legal, then `a2ps` is not the tool you need. Indeed `a2ps` just looks for some keywords, or some *sequences*, i.e., sequences of characters which are opened by a given marker, and ended the same way. For instance in `C`, the comments are opened by '`/*`', and ended by '`*/`'.

6.2 Known Style Sheets

`a2ps` version 4.9.9 supports the following languages: The current index (sum of all the style sheets version number) is 48.74 and the highest requirement is a2ps version 4.9.8. The known styles are:

68000 ('`68000.ssh`') Style Sheet
It requires a2ps version 4.9.7.

a2ps configuration file ('`a2psrc.ssh`') Style Sheet

a2ps style sheet ('`ssh.ssh`') Style Sheet
It requires a2ps version 4.9.5. -g substitutes the LaTeX symbols.

Ada ('`ada.ssh`') Style Sheet
It requires a2ps version 4.9.6.

Bourne Shell ('`sh.ssh`') Style Sheet
Some classical program names, or builtin, are highlighted in the second level of pretty-printing.

C ('`c.ssh`') Style Sheet
It requires a2ps version 4.9.7. This style does not high light the function definitions. Another style which high lights them, GNUish C, is provided (gnuc.ssh). It works only if you respect some syntactic conventions.

C Shell ('`csh.ssh`') Style Sheet
Written by Jim Diamond `jdiamond@fox.nstn.ca`. Some classical program names, and/or builtins, are highlighted in the second level of pretty-printing.

C++ ('`cpp.ssh`') Style Sheet
It inherits from: '`c.ssh`' .

CAML ('`caml.ssh`') Style Sheet
This style should also suit other versions of ML (caml light, SML etc.)

ChangeLog ('`chlog.ssh`') Style Sheet
This style covers the usual ChangeLog files.

Claire ('`claire.ssh`') Style Sheet
Claire is a high-level functional and object-oriented language with advanced rule processing capabilities. It is intended to allow the programmer to express complex algorithms with fewer lines and in an elegant and readable manner.

To provide a high degree of expressivity, CLAIRE uses:

- A very rich type system including type intervals and second-order types (with dual static/dynamic typing),
- Parametric classes and methods,
- An object-oriented logic with set extensions,
- Dynamic versioning that supports easy exploration of search spaces.

To achieve its goal of readability, CLAIRE uses

- set-based programming with an intuitive syntax,
- simple-minded object-oriented programming,
- truly polymorphic and parametric functional programming,
- a powerful-yet-readable extension of DATALOG to express logical conditions,
- an entity-relation approach with explicit relations, inverses, unknown values and relational
- operations.

More information on claire can be found on claire home page[1]

Common Lisp ('`clisp.ssh`') Style Sheet
Written by Juliusz Chroboczek `jec@dcs.ed.ac.uk`. It requires a2ps version 4.9.8.

Coq Vernacular ('`coqv.ssh`') Style Sheet
This style is devoted to the Coq v 5.10 vernacular language.

Eiffel ('`eiffel.ssh`') Style Sheet
It requires a2ps version 4.9.5.

Emacs Lisp ('`elisp.ssh`') Style Sheet

Encapsulated PostScript ('`eps.ssh`') Style Sheet
It requires a2ps version 4.9.8. It inherits from: '`ps.ssh`' . Illegal PostScript operators are highlighted as Errors.

[1] `http://www.ens.fr/~laburthe/claire.html`

Fortran (‘`fortran.ssh`’) Style Sheet
Written by Denis Girou `Denis.Girou@idris.fr`. It requires a2ps version 4.9.7.

GNUish C (‘`gnuc.ssh`’) Style Sheet
It requires a2ps version 4.9.8. It inherits from: ‘`c.ssh`’ . Declaration of functions are highlighted _only_ if you start the function name in the first column, and it is followed by an opening parenthesis. In other words, if you write

```
int main (void)
```

it won't work. Write:

```
int
main (void)
```

Java (‘`java.ssh`’) Style Sheet

LACE (‘`lace.ssh`’) Style Sheet
It requires a2ps version 4.9.5. This is meant for the Eiffel equivalent of the Makefiles.

Lex (‘`lex.ssh`’) Style Sheet
It requires a2ps version 4.9.7. It inherits from: ‘`c.ssh`’ . In addition to the C constructs, it highlights the declaration of states, and some special ‘%’ commands.

Mail Folder (‘`mail.ssh`’) Style Sheet
It requires a2ps version 4.9.7. To use from elm and others, it is better to specify -g -Email, since the file sent to printer is no longer truly a mailfolder. This style also suits to news. –strip options are also useful (they strip "useless" headers).

Whenever the changes of encoding are clear, a2ps sets itself the encoding for the parts concerned.

Tag 1 is the subject, and Tag 2 the author of the mail/news.

Makefile (‘`make.ssh`’) Style Sheet
It requires a2ps version 4.9.8. It inherits from: ‘`sh.ssh`’ . Special tokens, and non terminal declarations are highlighted.

Modula 2 (‘`modula2.ssh`’) Style Sheet
Written by Peter Bartke `bartke@inf.fu-berlin.de`. It requires a2ps version 4.9.7.

Modula 3 (‘`modula3.ssh`’) Style Sheet
It requires a2ps version 4.9.7.

o2c (‘`o2c.ssh`’) Style Sheet
It requires a2ps version 4.9.7. It inherits from: ‘`c.ssh`’ .

Oberon (‘`oberon.ssh`’) Style Sheet
It requires a2ps version 4.9.7.

Objective C (‘`objc.ssh`’) Style Sheet
Written by Paul Shum `pshum@ali.bc.ca`. It requires a2ps version 4.9.7. It inherits from: ‘`c.ssh`’ .

Octave/MATLAB (‘`octave.ssh`’) Style Sheet
Written by C.P. Earls. It requires a2ps version 4.9.5.

Oracle parameter file (‘`initora.ssh`’) Style Sheet
Written by Pierre Mareschal `pmaresch@be.oracle.com`. It requires a2ps version 4.9.5. For init.ora parameter files.

Oracle PL/SQL (‘`plsql.ssh`’) Style Sheet
Written by Pierre Mareschal `pmaresch@be.oracle.com`. This style is to be checked.

Oracle SQL (‘`sql.ssh`’) Style Sheet
Written by Pierre Mareschal `pmaresch@be.oracle.com`. a2ps-sql Pretty Printer Version 1.0.0 beta - 18-MAR-97 For comments, support for – /*..*/ and //. This style is to be checked.

Oracle SQL-PL/SQL-SQL*Plus (‘`oracle.ssh`’) Style Sheet
Written by Pierre Mareschal `pmaresch@be.oracle.com`. 18-MAR-97 For comments, support for – /*..*/ and //. This style is to be checked.

Pascal (‘`pascal.ssh`’) Style Sheet
It requires a2ps version 4.9.7. The standard Pascal is covered by this style. But some extension have been added too, hence modern Pascal programs should be correctly handled.

Perl (‘`perl.ssh`’) Style Sheet
Written by Denis Girou `Denis.Girou@idris.fr`.

PostScript (‘`ps.ssh`’) Style Sheet
It requires a2ps version 4.9.7. Only some keywords are high lighted, because otherwise listings are quickly becoming a big bold spot.

PostScript Printer Description (‘`ppd.ssh`’) Style Sheet
It requires a2ps version 4.9.8. The support is not yet complete. Tag 1 is the printer ModelName.

PreScript (‘`pre.ssh`’) Style Sheet
It requires a2ps version 4.9.7. This is prescript, a language of description of textual pages. It provides by the use of LaTeX like commands, a way to describe the pages that this program should produce. It can be a very good choice of destination language for people who want to produce text to print (e.g. pretty-printing, automating documentation etc.) but who definitely do not want to learn postscript, nor to require the use of LaTeX.

Prolog (‘`prolog.ssh`’) Style Sheet
Under construction.

Promela (‘promela.ssh’) Style Sheet
There is no way for this program to highlight send and receive primitives.

Python (‘python.ssh’) Style Sheet
It requires a2ps version 4.9.5.

Sather (‘sather.ssh’) Style Sheet
It requires a2ps version 4.9.5.

Scheme (‘scheme.ssh’) Style Sheet

SDL-88 (‘sdl88.ssh’) Style Sheet
Written by Jean-Philippe Cottin cottin@inf.enst.fr. –strip-level=2 is very useful: it cancels the graphical information left by graphic editors. Only the pure specification is then printed.

SQL 92 (‘sql92.ssh’) Style Sheet
Written by Pierre Mareschal pmaresch@be.oracle.com. 18-MAR-97 This style is to be checked.

tcsh (‘tcsh.ssh’) Style Sheet
Written by Jim Diamond jdiamond@fox.nstn.ca. It requires a2ps version 4.9.7. It inherits from: ‘csh.ssh’ .

TeX (‘tex.ssh’) Style Sheet
Written by Denis Girou Denis.Girou@idris.fr. It requires a2ps version 4.9.7. This is the style for (La)TeX files. It’s mainly useful for people who develop (La)TeX packages. With -g, common mathematical symbols are represented graphically.

Texinfo (‘texinfo.ssh’) Style Sheet
It requires a2ps version 4.9.7. Definitely being built. With the option -g, the nodes will be printed on separated pages which title is the name of the node.

tk (‘tk.ssh’) Style Sheet
It inherits from: ‘tcl.ssh’ . Since everything, or almost, is a string, what is printed is not always what you would like.

Tool Command Language (‘tcl.ssh’) Style Sheet
Since everything, or almost, is a string, what is printed is not always what you would like.

Unity (‘unity.ssh’) Style Sheet
Written by Jean-Philippe Cottin cottin@inf.enst.fr. The graphic conversion of the symbols (option -g) is nice.

VERILOG (‘verilog.ssh’) Style Sheet
Written by Edward Arthur eda@ultranet.com. It requires a2ps version 4.9.7. This style is devoted to the VERILOG hardware description language.

VHDL (‘vhdl.ssh’) Style Sheet
Written by Thomas Parmelan Thomas.Parmelan@efrei.fr. This experimental style is devoted to the VHDL modelisation language.

VRML (‘vrml.ssh’) Style Sheet
Written by Nadine Richard Nadine.Richard@ada.eu.org. It requires a2ps version 4.9.5. According to Grammar Definition Version 2.0 ISO/IEC CD 14772[2]

Yacc (‘yacc.ssh’) Style Sheet
It requires a2ps version 4.9.8. It inherits from: ‘c.ssh’ . Special tokens, and non terminal declarations are highlighted.

zsh (‘zsh.ssh’) Style Sheet

6.3 Faces

A *face* is an attribute given to a piece of text, which specifies how it should look like. Since a2ps is devoted to pretty-printing source files, the faces it uses are related to the syntactic entities that can be encountered in a file.

The faces a2ps uses are:

‘Plain’ This corresponds to the text body.

‘Keyword’
‘Keyword_strong’
These are related to the keywords that may appear in a text.

‘Comment’
‘Comment_strong’
These are related to comments in the text. Remember that comments should be considered as non essential ("*Aaaeaaarg*" says the programmer); indeed, the user might suppress the comments thanks (?) to the option ‘--strip-level’. Hence, **never** use these faces just because you think they look better on, say, strings.

‘Label’
‘Label_strong’
These are used when a point of extreme importance, or a sectioning point, is met. Typically, functions declarations etc.

‘String’ Used mainly for string and character literals.

‘Error’ Used to underline the presence of an error. For instance in Encapsulated PostScript, some PostScript operators are forbidden: they are underlined as errors.

Actually, there is also the face ‘Symbol’, but this one is particular: it is not legal changing its font.

[2] http://vag.vrml.org/VRML2.0/FINAL/spec/part1/grammar.html

6.4 Style sheets semantics

`a2ps` pretty prints a source file thanks to *style sheets*, one per language. In the following is described how the style sheets are defined. You may skip this section if you don't care how `a2ps` does this, and if you don't expect to implement new styles.

6.4.1 Name and key

Every style sheet has both a key, and a name. The name can be clean and beautiful, with any character you might want. The key is in fact the prefix part of the file name, and is alpha-numerical, lower case, and less than 8 characters long.

Anywhere `a2ps` needs to recognize a style sheet by a name, **it uses the key** (in the ‘`sheets.map`’ file, with the option ‘`-E`’, etc.).

As an example, C++ is implemented in a file called ‘`cpp.ssh`’, in which the name is declared to be ‘`C++`’.

The rationale is that not every system accepts any character in the file name (e.g., no ‘`+`’ in MS-DOS). Moreover, it allows to make symbolic links on the ssh files (e.g., ‘`ln -s cpp.ssh c++.ssh`’ let's you use ‘`-E c++`’).

6.4.2 Comments

ssh files can include the name of its author, a version number, a documentation note and a requirement on the version of `a2ps`. For instance, if a style sheet requires `a2ps` version 4.9.6, then `a2ps` version 4.9.5 will reject it.

6.4.3 Alphabets

`a2ps` needs to know the beginning and the end of a word, especially keywords. Hence it needs two alphabets: the first one specifying by which letters an identifier can begin, and the second one for the rest of the word. If you prefer, a keyword starts with a character belonging to the first alphabet, and a character not pertaining to the second is a separator.

6.4.4 Case sensitivity

If the style is case insensitive, then matching is case insensitive (keywords, operators and sequences).

6.4.5 Symbols

A *symbol* is a structure which consists of two items:

1. its source string, with which the source file is compared;
2. a list of faced strings which will replace the text matched in the pretty-printed output. A faced string is composed of
   - a string, or a reference to a part of the source string (see section “Back-reference Operator” in *Regex manual*)
   - the face to use to print it

Just a short example: ‘`(foo, bar, Keyword_strong)`’ as a symbol means that every input occurrence of ‘`foo`’ will be replaced by ‘`bar`’, written with the `Keyword_strong` face.

If the destination string is empty, then `a2ps` will use the source string. This is different from giving the source string as a destination string if the case is different. An example will make it fairly clear.

Let `foobar` be a case insensitive style sheet including the symbols ‘`(foo, "", Keyword)`’ and ‘`(bar, bar, Keyword)`’. Then, on the input ‘`FOO BAR`’, `a2ps` will produce ‘`FOO bar`’ in `Keyword`.

`a2ps` implements two different ways to match a string. The difference comes from that some keywords are sensitive to the delimiters around them (such as ‘`unsigned`’ and ‘`int`’ in `C`, which are definitely not the same thing as ‘`unsignedint`’), and others not (in `C`, ‘`!=`’ is "different from" both in ‘`a != b`’ and ‘`a!=b`’).

The first ones are called *keywords* in `a2ps` jargon, and the seconds are *operators*. Operators are matched any where they appears, while keywords need to have separators around them (see Section 6.4.3 [Alphabets], page 47).

Let us give a more complicated example: that of the `Yacc` rules. A rule in `Yacc` is of the form:

```
a_rule : part1 part2 ;
```

Suppose you want to highlight these rules. To recognize them, you will write a regular expression specifying that:

1. it must start at the beginning of the line,
2. then there is string composed of symbols, which is what you want to highlight,
3. and a colon, which can be preceded by blank characters.

The regexp you want is: ‘`/^[a-zA-Z0-9_]*[\t ]*:`’. But if you use the symbol ‘`/^[a-zA-Z0-9_]*[\t ]*:, "", Label_strong`’, then the blanks and the colon are highlighted too. Hence you need to specify some parts in the regexp (see section “Back-reference Operator” in *Regex manual*), and use a longer list of destination strings. The correct symbol is

```
(/^\\([a-zA-Z0-9_]*\\)\\([\t ]*:\\)/, \1 Label_strong, \2 Plain)
```

6.4.6 Sequences

A *sequence* is a string between two *markers*, along with a list of exceptions. A marker is a fixed string. Typical examples are comments, string (with usually ‘`"`’ as opening and closing markers, and ‘`\\`’ and ‘`\"`’ as exceptions) etc. Three faces are used: one for the initial marker, one for the core of the sequence, and a last one for the final maker.

6.4.7 Optional entries

There are two levels of pretty-printing encoded in the style sheets. By default, `a2ps` uses the first level, unless the option ‘`-g`’ is specified, in which case, optional keywords, operators and sequences are considered.

6.5 Style sheets implementation

6.5.1 A bit of syntax

Here are the lexical rules underlying the style sheet language:

- the separators are white space, form feed, new line, and tab.
- '`#`' introduces a comment, ended at the end of the line.
- special characters are the separators, plus '`#`', '`"`', '`,`', '`(`', '`)`', '`+`' and '`/`'. Any other character is a regular character.
- the list of the structuring keywords is

  `alphabet`, `alphabets`, `are`, `case`, `documentation`, `end`, `exceptions`, `first`, `in`, `insensitive`, `is`, `keywords`, `operators`, `optional`, `second`, `sensitive`, `sequences`, `style`
- the list of the keywords designating faces is

  `Comment`, `Comment_strong`, `Encoding`, `Error`, `Index1`, `Index2`, `Index3`, `Index4`, `Invisible`, `Keyword`, `Keyword_strong`, `Label`, `Label_strong`, `Plain`, `String`, `Symbol`, `Tag1`, `Tag2`, `Tag3`, `Tag4`
- the list of keywords designating special sequences is

  `C-char`, `C-string`
- the list of keywords representing special characters is

  `---`, `\Alpha`, `\Beta`, `\Chi`, `\Delta`, `\Downarrow`, `\Epsilon`, `\Eta`, `\Gamma`, `\Im`, `\Iota`, `\Kappa`, `\Lambda`, `\Leftarrow`, `\Leftrightarrow`, `\Mu`, `\Nu`, `\Omega`, `\Omicron`, `\Phi`, `\Pi`, `\Psi`, `\Re`, `\Rho`, `\Rightarrow`, `\Sigma`, `\Tau`, `\Theta`, `\Uparrow`, `\Upsilon`, `\Xi`, `\Zeta`, `\aleph`, `\alpha`, `\angle`, `\approx`, `\beta`, `\bullet`, `\cap`, `\carriagereturn`, `\cdot`, `\chi`, `\circ`, `\clubsuit`, `\cong`, `\copyright`, `\cup`, `\delta`, `\diamondsuit`, `\div`, `\downarrow`, `\emptyset`, `\epsilon`, `\equiv`, `\eta`, `\exists`, `\florin`, `\forall`, `\gamma`, `\geq`, `\heartsuit`, `\in`, `\infty`, `\int`, `\iota`, `\kappa`, `\lambda`, `\langle`, `\lceil`, `\ldots`, `\leftarrow`, `\leftrightarrow`, `\leq`, `\lfloor`, `\mu`, `\nabla`, `\neq`, `\not`, `\not\in`, `\not\subset`, `\nu`, `\omega`, `\omicron`, `\oplus`, `\otimes`, `\partial`, `\perp`, `\phi`, `\pi`, `\pm`, `\prime`, `\prod`, `\propto`, `\psi`, `\radicalex`, `\rangle`, `\rceil`, `\register`, `\rfloor`, `\rho`, `\rightarrow`, `\sigma`, `\sim`, `\spadesuit`, `\subset`, `\subseteq`, `\suchthat`, `\sum`, `\supset`, `\supseteq`, `\surd`, `\tau`, `\theta`, `\therefore`, `\times`, `\trademark`, `\uparrow`, `\upsilon`, `\varUpsilon`, `\varcopyright`, `\vardiamondsuit`, `\varphi`, `\varpi`, `\varregister`, `\varsigma`, `\vartheta`, `\vartrademark`, `\vee`, `\wedge`, `\wp`, `\xi`, `\zeta`
- a string starts and finishes with '`"`', and may contain anything. Regular `C` escaping mechanism is used.
- any sequence of regular characters which is not a keyword, is a string (consider this as a shortcut, avoiding extraneous '`"`').

6.5.2 Style sheet header

The definition of the name of the style sheet is:


```
style name is
  # body of the style sheet
end style
```

The following constructions are optional:

`version`
: To define the version number of the style sheet

  `version is` *version-number*

`written`
: To define the author(s).

  `written by` *author(s)-name(s)*

  Giving your email is useful for bug reports about style sheets.

  `written by "Some Body <Some.Body@some.whe.re>"`

`requires`
: To specify the version of `a2ps` it requires. `a2ps` won't accept a file which requires a higher version number than its own.

  `requires a2ps` *a2ps-version-number*

`documentation`
: To leave extra comments people should read.

  ```
  documentation is
    strings
  end documentation
  ```

  strings may be a list of strings, without comas, in which case new lines are automatically inserted between each item.

  Please, write useful comments, not '`This style is devoted to C files`', since the name is here for that, nor '`Report errors to mail@me.somewhere`', since `written by` is there for that.

  ```
  documentation is
    "Not all the keywords are used, to avoid too much bolding."
    "Second level of pretty-printing (-g), covers the whole language."
  end documentation
  ```

6.5.3 Syntax of the Words

There are two things `a2ps` needs to know: what is symbol consistent, and whether the style is case insensitive.

`alphabet`
: To define two different alphabets, use

  `first alphabet is` string
  `second alphabet is` string

  If both are identical, you may use the shortcut

  `alphabets are` string

  The default alphabets are

```
first alphabet is
  "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ_"
second alphabet is
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ_\
0123456789"
```

Note that it is on purpose that no characters interval are used.

`case`

```
case insensitive        # e.g., C, C++ etc.
case sensitive          # e.g., Perl, Sather, Java etc.
```

The default is `case insensitive`.

6.5.4 Inheriting from Other Style Sheets

It is possible to extend an existing style. The syntax is:

```
ancestors are
  ancestor1[, ancestor2...]
end ancestors
```

where *ancestor1* etc. are style sheet keys.

For semantics, the rules are the following:

- the ancestors are read in order;
- the definition of the current style is read last;
- the last defined alphabets are those used;
- the case sensitivity is the last read.
- when there are several definitions of the same keywords or operators or sequences *defined as strings* (not regular expressions), then the last definition is winning;
- definitions based on regular expression are just concatenated, the last included being the first checked.

As an example, both `C++` and `Objective C` style sheets extend the `C` style sheet:

```
style "Objective C" is
#[...]
ancestors are
   c
end ancestors
#[...]
end style
```

To the biggest surprise of the author, mutually dependent style sheets do work!

6.5.5 Syntax for the symbols

See Section 6.4.5 [Symbols], page 47, for the definition of *symbol*.

Because of various short cuts, there are many ways to declare a symbol:


```
symbol      := matcher destination_and_face
            | ( matcher faced_destination_list ) ;
faced_destination_list := faced_destination , ...
faced_destination :=
              latex-keyword
            | destination face
matcher     := string | regex ;
destination := string | \num | <nothing>;
face        := face-keyword | <nothing>;
```

The rules are the following:

- If the matcher is a regular expression, then they are compiled with the following syntax bits:

  ```
  #define RE_SYNTAX_A2PS \
     (RE_SYNTAX_EMACS | RE_CHAR_CLASSES | RE_INTERVALS)
  ```

  See section "Regular Expression Syntax" in *Regex manual*, for detailed description of the regular expressions used.
- If no *destination* is specified, then the string matched in the source is used. For instance '`(/fo*/, NULL, Keyword)`' applied on the source '`"fooooo"`' produces '`"fooooo"`' in `Keyword`.
- If no *face* is given, then
  - if the context defines the default face, then this face is used;
  - if no default face is given, `PLAIN` is used.

6.5.6 Declaring the keywords and the operators

Basically, keywords and operators are lists of symbols. The syntax is:

```
keywords are
  symbol1, symbol2, ... symboln
end keywords
```

or

```
keywords in face-keyword are
  symbol1, symbol2, ... symboln
end keywords
```

is which case the default face is set to *face-keyword*.

As an example:

```
keywords in Keyword_strong are
  /foo*/,
  "bar" "BAR" Keyword,
  -> \rightarrow
end keywords
```

is valid.

The syntax for the operators is the same, and both constructs can be qualified with an `optional` flag, in which case they are not taken into account unless option '`-g`' is given (see Section 3.1.6 [Pretty Print Options], page 18).

This is an extract of the `C` style sheet:

```
optional operators are
   -> \rightarrow,
   && \wedge,
   || \vee,
   != \neq,
   == \equiv,
   # We need to protect these, so that <= is not replaced in <<=
   <<=,
   >>=,
   <= \leq,
   >= \geq,
   ! \not
end operators
```

Note how '`<<=`' and '`>>=`' are protected (there are defined to be written as is when met in the source). This is to prevent the two last characters of '`<<=`' from being converted into a 'less or equal' sign.

The order in which you define the elements of a category (but the sequences) does not matter. But since `a2ps` sorts them at run time, it may save time if the alphabetical `C`-order is more or less followed.

You should be aware that when declaring a keyword with a regular expression as matcher, then `a2ps` automatically makes this expression matching only if there are no character of the first alphabet both just before, and just after the string.

In term of implementation, it means that

```
keywords are
  /foo\\|bar/
end keywords
```

is exactly the same as

```
operators are
  /\\b\\(foo\\|bar\\)\\b/
end operators
```

This can cause problems if you use anchors (e.g. `$`, or `^`) in keywords: the matcher will be broken. In this particular case, define your keywords as operators, taking care of the '`\\b`' by yourself.

See section "Match-word-boundary Operator" in *Regex manual*, for details on '`\b`'.

6.5.7 Declaring the sequences

Sequences admit several declarations too:

```
sequences      := sequences are
                   sequence1, sequence2, ... sequencen
                  end sequences
sequence       := open in-face close_opt exceptions_opt
                | C-string
                | C-char
                ;
open           := symbol ;
```



```
close_opt      := symbol
                | closers are
                    list-of-symbols
                  end closers
                | <nothing>
                ;
exceptions_opt := exceptions are
                    list-of-symbols
                  end exceptions
                | <nothing>
                ;
```

The rules are:

- The default face is then *in-face*.
- If no closing symbol is given, '`/$/`' (i.e., end-of-line) is used.

As a first example, the definition of `C-string` is:

```
sequences are
 "\"" Plain String "\"" Plain
     exceptions are
        "\\\\", "\\\""
     end exceptions
end sequences
```

The following example comes from '`ssh.ssh`', the style sheet for style sheet files, in which there are two kinds of pseudo-strings: the strings ('`"example"`'), and the regular expressions ('`/example/`'). We do not want the content of the pseudo-strings in the face `String`.

```
sequences are
  # The comments
  "#" Comment,

  # The name of the style sheet
  "style " Keyword_strong (Label + Decl1) " is" Keyword_strong,

  # Strings are exactly the C-strings, though we don't want to
  # have them in the "string" face
  "\"" Plain "\""
     exceptions are
        "\\\\", "\\\""
     end exceptions,

  # Regexps
  "/" Plain "/"
     exceptions are
        "\\\\", "\\\/"
     end exceptions

end sequences
```

The order between sequences does matter. For instance in java, '/**' introduces strong comments, and '/*' comments. '/**' *must* be declared before '/*', or it will be hidden.

There are actually some sequences that could have been implemented as operators with a specific regular expression (that goes up to the closer). Nevertheless be aware of a big difference: regular expression are applied to a single line of the source file, hence, they cannot match on several lines. For instance, the C comments,

```
/*
 * a comment
 */
```

cannot be implemented with operators, though C++ comments can:

```
//
// a comment
//
```

6.6 A tutorial on style sheets

In this section a simple example of style sheet is entirely covered: that of 'ChangeLog' files.

'ChangeLog' files are some kind of memory of changes done to files, so that various programmers can understand what happened to the sources. This helps a lot, for instance, in guessing what recent changes may have introduced new bugs.

6.6.1 Example and syntax

First of all, here is a sample of a 'ChangeLog' file, taken from the 'misc/' directory of the original a2ps package:

```
Sun Apr 27 14:29:22 1997  Akim Demaille  <demaille@inf.enst.fr>

        * base.ps: Merged in color.ps, since now a lot is
          common [added box and underline features].

Fri Apr 25 14:05:20 1997  Akim Demaille  <demaille@inf.enst.fr>

        * color.ps: Added box and underline routines.

Mon Mar 17 20:39:11 1997  Akim Demaille  <demaille@gargantua.enst.fr>

        * base.ps: Got rid of CourierBack and reencoded_backspace_font.
          Now the C has to handle this by itself.

Sat Mar  1 19:12:22 1997  Akim Demaille  <demaille@gargantua.enst.fr>

        * *.enc: they build their own dictionaries, to ease multi
          lingual documents.
```

The syntax is really simple: A line specifying the author and the date of the changes, then a list of changes, all of them starting with an asterix followed by the name of the files concerned, then optionally between parentheses the functions affected, and then some comments.

6.6.2 Implementation

Quite naturally the style will be called ChangeLog, hence:

```
style ChangeLog is
written by "Akim Demaille <demaille@inf.enst.fr>"
version is 1.0
requires a2ps 4.9.5

documentation is
   "This is a tutorial style sheet.\n"
end documentation
  ...
end style
```

A first interesting and easy entry is that of function names, between '(' and ')':

```
sequences are
  "(" Plain Label ")" Plain
end sequences
```

A small problem that may occur is that there can be several functions mentioned separated by commas, that we don't want to highlight this way. Commas, here, are exceptions. Since regular expressions are not yet implemented in a2ps, there is a simple but stupid way to avoid that white spaces are all considered as part of a function name, namely defining two exceptions: one which captures a single comma, and a second, capturing a comma and its trailing space.

For the file names, the problem is a bit more delicate, since they may end with ':', or when starts the list of functions. Then, we define two sequences, each one with one of the possible closers, the exceptions being attached to the first one:

```
sequences are
  "* " Plain Label_strong ":" Plain
     exceptions are
        ", " Plain, "," Plain
     end exceptions,
  "* " Plain Label_strong " " Plain
end sequences
```

Finally, let us say that some words have a higher importance in the core of text: those about removing or adding something.

```
keywords in Keyword_strong are
  add, added, remove, removed
end keywords
```

Since they may appear in lower or upper, of mixed case, the style will be defined as case insensitive.

Finally, we end up with this style sheet file, in which an optional highlighting of the mail address of the author is done. Saving the file is last step. But do not forget that a style sheet has both a name as nice as you may want (such as '`Common Lisp`'), and a key on which there are strict rules: the prefix must be alpha-numerical, lower case, with no more than 8 characters. Let's chose '`chlog.ssh`'.

```
# This is a tutorial on a2ps' style sheets
style ChangeLog is
written by "Akim Demaille <demaille@inf.enst.fr>"
version is 1.0
requires a2ps 4.9.5

documentation is
   "Second level of high lighting covers emails."
end documentation

sequences are
  "(" Plain Label ")" Plain
     exceptions are
        ", " Plain, "," Plain
     end exceptions,
  "* " Plain Label_strong ":" Plain
     exceptions are
        ", " Plain, "," Plain
     end exceptions,
  "* " Plain Label_strong " " Plain
end sequences

keywords in Keyword_strong are
  add, added, remove, removed
end keywords

optional sequences are
   < Plain Keyword > Plain
end sequences
end style
```

6.6.3 The Entry in '`sheets.map`'

The last touch is to include the pattern rules about '`ChangeLog`' files (which could appear as '`ChangeLog.old`' etc.) in '`sheets.map`':

```
# ChangeLog files
ChangeLog*                  chlog
```

This won't work... Well, not always. Not for instance if you print '`misc/ChangeLog`'. This is not a bug, but truly a feature, since sometimes one gets more information about the type of a file from its path, than from the file name.

Here, to match the preceding path that may appear, just use '`*`':

```
# ChangeLog files
*ChangeLog*                 chlog
```

If you want to be more specific ('`FooChangeLog`' should not match), use:


```
# ChangeLog files
ChangeLog*                  chlog
*/ChangeLog*                chlog
```

6.6.4 More Sophisticated Rules

The example we have presented until now uses only basic features, and does not take advantage of the regexp. In this section we should how to write more evolved pretty printing rules.

The target will be the lines like:

```
Sun Apr 27 14:29:22 1997  Akim Demaille  <demaille@inf.enst.fr>

Fri Apr 25 14:05:20 1997  Akim Demaille  <demaille@inf.enst.fr>
```

There are three fields: the date, the name, the mail. These lines all start at the beginning of line. The last field is the easier to recognize: is starts with a '`<`', and finishes with a '`>`'. Its rule is then '`/<[^>]+>/`'. It is now easier to specify the second: it is composed only of words, at least one, separated by blanks, and is followed by the mail: '`/[[:alpha:]]+\\([ \t]+[[:alpha:]]+\\)*/`'. To concatenate the two, we introduce optional blanks, and we put each one into a pair of '`\\(`'-'`\\)`' to make each one a recognizable part:

```
\\([[:alpha:]]+\\([ \t]+[[:alpha:]]+\\)*\\)
\\(.+\\)
\\(<[^>]+>\\)
```

Now the first part is rather easy: it starts at the beginning of the line, finishes with a digit. Once again, it is separated from the following field by blanks. Split by groups (see section "Grouping Operators" in *Regex manual*), we have:

```
^
\\([^\t ].*[0-9]\\)
\\([ \t]+\\)
\\([[:alpha:]]+\\([ \t]+[[:alpha:]]+\\)*\\)
\\(.+\\)
\\(<[^>]+>\\)
```

Now the destination is composed of back references to those groups, together with a face[3]:

```
# We want to highlight the date and the maintainer name
optional operators are
  (/^\\([^\t ].*[0-9]\\)
      \\([ \t]+\\)
      \\([[:alpha:]]+\\([ \t]+[[:alpha:]]+\\)*\\)
      \\(.+\\)
      \\(<[^>]+>\\)/
   \1 Keyword, \2 Plain, \3 Keyword_strong, \5 Plain, \6 Keyword)
end operators
```

[3] In the real style sheet file, the regex must not be split upon several lines.

7 PostScript

This chapter is devoted to the information which is only relevant to PostScript.

7.1 Page Device Options

Page device is a PostScript level 2 feature that offers an uniform interface to control printer's output device. a2ps protects all page device options inside an if block so they have no effect in level 1 interpreters. Although all level 2 interpreters support page device, they do not have to support all page device options. For example some printers can print in duplex mode and some can not. Refer to the documentation of your printer for supported options.

Here are some usable page device options which can be selected with the '`-D`' option ('`--setpagedevice`'). For a complete listing, see *PostScript Language Reference Manual*: section 4.11 Device Setup.

`Collate` *boolean*
: how output is organized when printing multiple copies

`Duplex` *boolean*
: duplex (two side) printing

`ManualFeed` *boolean*
: manual feed paper tray

`OutputFaceUp` *boolean*
: print output 'face up' or 'face down'

`Tumble` *boolean*
: how opposite sides are positioned in duplex printing

7.2 Statusdict Options

The `statusdict` is a special storage entity in PostScript (called a *dictionary*), in which some variables and operators determine the behavior of the printer. This is an historic horror that existed before page device definitions were defined. They are even more printer dependent, and are provided only for the people who don't have a level printer. In any case, refer to the documentation of your printer for supported options.

Here are some statusdict definitions in which you might be interested:

`manualfeed` *boolean*
: Variable which determine that the manual fed paper tray will be used. Use is '`-Smanualfeed::true`'.

`setmanualfeed` *boolean*
: Idem as the previous point, but use is '`-Ssetmanualfeed:true`'.

`setduplexmode` *boolean*
: If *boolean*, then print in Duplex mode. Use if '`-Ssetduplexmode:true`'.

7.3 Colors in PostScript

Nevertheless, here are some tips on how to design your PostScript styles. It is strongly recommended to use '`gray.pro`' or '`color.pro`' as a template.

There are two PostScript instructions you might want to use in your new PostScript prologue:

`setgray`
: this instruction must be preceded by a number between 0 (black) and 1 (white). It defines the gray level used.

`setrgbcolor`
: this instruction must be preceded by three numbers between 0 (0 %) and 1 (100%). Those three numbers are related to red, green and blue proportions used to designate a color.

`a2ps` uses two upper level procedures, `BG` and `FG`, but both use an argument as in `setrgbcolor`. So if you wanted a gray shade, just give three times the same ratio.

7.4 Designing PostScript prologues

It is pretty known that satisfying the various human tastes is an NEXPTIME-hard problem, so `a2ps` offers ways to customize its output through the *prologue files*. But since the authors feel a little small against NEXPTIME, they agreed on the fact that **you** are the one who will design the look you like.

Hence in this section, you will find what you need to know to be able to customize a2ps output.

Basically, `a2ps` uses *faces* which are associated to their "meaning" in the text. `a2ps` let's you change the way the faces look.

7.4.1 Definition of the faces

There are three things that define a face:

Its font
: You should never call the font by yourself, because sometimes `a2ps` may decide that another font would be better. This is what happens for instance if a font does not support the encoding you use.

  Hence, never set the font by yourself, but ask `a2ps` to do it. This is done through a line:

  `%Face:` *face real-font-name size*

  This line tells `a2ps` that the font of *face* is *real-font-name*. It will replace this line by the correct PostScript line to call the needed font, and will do everything needed to set up the font.

  The size of the text body is `bfs`.

Its background color
: There are two cases:

  1. You want a background color, then give the *RGB* (see Section 7.3 [Colors in PostScript], page 60) ratio and `true` to `BG`:

```
0.8 0.8 0 true BG
```

2. You don't want a background color, then call `BG` with `false`:

```
false BG
```

Its foreground color
As `BG`, call `FG` with an *RGB* ratio:

```
0 0.5 0 FG
```

Its underlining
`UL` requires a boolean argument, depending whether you want or not the current face to be underlined.

```
true UL
```

Its boxing Requiring a boolean, `BX` let's a face have a box drawn around.

7.4.2 A step by step example

We strongly suggest our readers not to start from scratch, but to copy one of the available styles (see the result of '`a2ps --list-features`'), to drop it in one of `a2ps` directories (say '`$HOME/.a2ps`', and to patch it until you like it.

Here, we will start from '`color.pro`', trying to give it a funky look.

Say you want the keywords to be in Helvetica, drawn in a flashy pink on a light green. And strong keywords, in Times Bold Italic in brown on a soft Hawaïan sea green (you are definitely a fine art *amateur*).

Then you need to look for '`k`' and '`K`':

```
/k {
  false BG
  0 0 0.9 FG
%Face: Keyword Courier bfs
  Show
} bind def

/K {
  false BG
  0 0 0.8 FG
%Face: Keyword_strong Courier-Bold bfs
  Show
} bind def
```

and turn it into:

```
/k {
  0.2 1 0.2 true BG
  1 0.2 1 FG
%Face: Keyword Helvetica bfs
  Show
} bind def

/K {
```



```
  0.4 0.2 0 true BG
  0.5 1 1 FG
%Face: Keyword_strong Times_BoldItalic bfs
  Show
} bind def
```

Waouh! It looks great!

A bit trickier: let change the way the line numbers are printed.

First, let's look for the font definition:

```
%%BeginSetup
% The font for line numbering
/f# /Helvetica findfont bfs .6 mul scalefont def
%%EndSetup
```

Let it be in Times, twice bigger than the body font.

```
%%BeginSetup
% The font for line numbering
/f# /Times findfont bfs 2 mul scalefont def
%%EndSetup
```

How about its foreground color?

```
% Function print line number (<string> # -)
/# {
  gsave
    sx cw mul 2 div neg 0 rmoveto
    f# setfont
    0.8 0.1 0.1 FG
    c-show
  grestore
} bind def
```

Let it be blue. Now you know the process: just put '`0 0 1`' as `FG` arguments.

8 PreScript

`PreScript` has been designed in conjunction with `a2ps`. Since bold sequences, special characters etc. were implemented in `a2ps`, we thought it would be good to allow direct access to those features: `PreScript` became an input language for `a2ps`, where special font treatments are specified in a LaTeX-like syntax.

The main advantages for using `PreScript` are:

- it is fairly simple,
- it is LaTeX compatible (that is, correctly written `PreScript` files do compile with LaTeX)
- `a2ps` is small and easy to install, hence it is available on every UNIX platform.

It can be a good candidate for generation of PostScript output (syntactic pretty-printers, generation of various reports etc.).

8.1 Syntax

Every command name begins with a backslash ('`\`'). If the command uses an argument, it is given between curly braces with no spaces between the command name and the argument.

The main limit on `PreScript` is that no command can be used inside another command. For instance, though it is perfectly legal in LaTeX, the following line will be badly interpreted by `a2ps`:

```
\textbf{Problems using \textit{PreScript} instead of \LaTeX}
```

The correct way to write this in `PreScript` is

```
\textbf{Problems using} \textbi{PreScript} \textbf{instead of} \LaTeX.
```

Everything from an unquoted % to the end of line is ignored (comments).

8.2 `PreScript` commands

8.2.1 Structure commands

These commands required arguments.

'`\section{`*Title*`}`'
'`\subsection{`*Title*`}`'
'`\subsubsection{`*Title*`}`.'
: Used to specify the title of a section, subsection or subsubsection.

'`\textbf{`*text*`}`'
'`\textit{`*text*`}`'
'`\textbi{`*text*`}`'
'`\textrm{`*text*`}`'
: write *text* in bold, italic, bold-italic, Times. Default font is Courier. These instructions are said to be hard-coded. They may be removed in a future definition of PreScript. The following instructions are prefered.

'`\keyword{`*text*`}`'
'`\Keyword{`*text*`}`'
: Highlight lightly/strongly the given *text*. Should be used only for a couple of adjacent words.

'`\comment{`*text*`}`'
'`\Comment{`*text*`}`'
: The *text* is given a special face. The *text* may be removed if option '`--strip`' is used.

'`\label{`*text*`}`'
'`\Label{`*text*`}`'
: *text* should be considered as a definition, or an important point in the structure of the whole text.

'`\string{`*text*`}`'
: Write *text* with string's face (e.g., in font Times).

'`\textsy{`*text*`}`'
: *text* is written in the PostScript symbol font. This feature is not compatible with LaTeX. It is recommended, when possible, to use the special keywords denoting symbols, which are compatible with LaTeX (see Section 8.2.2 [Printed symbols], page 64).

'`\header{`*text*`}`'
'`\footer{`*text*`}`'
: Use *text* as header (footer) for the current page. If several headers or footers are defined on the same page, the last one is taken into account.

8.2.2 Printed symbols

There is a big menagerie of symbols. To get a list, it is suggested to print the style sheet file of `PreScript`:

```
a2ps -g pre.ssh
```

The reader should be aware that LaTeX is more demanding about special symbols. Most of them must be in so-called math mode, which means that the command must be inside $ signs. For instance, though

```
If \forall x \in E, x \in F then E \subseteq F.
```

is perfectly legal in `PreScript`, it should be written

```
If $\forall x \in E, x \in F$ then $E \subseteq F$.
```

for LaTeX. Since in `PreScript` every '`$`' is discarded (unless quoted by a '`\`'), the second form is also admitted.

8.2.3 Special characters

'`$`' is ignored in `PreScript` for compatibility with LaTeX, and '`%`' introduces a comment. Hence they are the only symbols which have to be quoted by a '`\`'. The following characters should also be quoted to produce good LaTeX files, but are accepted by `PreScript`: '`_`', '`&`', '`#`'.

Note that *inside a command*, like `\textbf`, the quotation mechanism does not work in `PreScript` (`\textrm{#$%}` writes '`#$%`') though LaTeX still requires quotation. Hence whenever special characters or symbols are introduced, they should be at the outer most level.

8.2.4 Quotation

There is a general mechanism to avoid special sequences from being interpreted: the commands '`\verb+`*quoted string*`+`', where '`+`' can be any symbol in '`+`', '`!`', '`|`', '`#`', '`=`'.

8.3 Differences with LaTeX

8.3.1 LaTeX commands ignored in PreScript

Some LaTeX commands have no equivalent in `PreScript`, but are simply skipped at compilation time (if immediately followed by an end-of-line):

- '`\begin{document}`'
- '`\end{document}`'
- '`\begin{itemize}`'
- '`\end{itemize}`'
- '`\begin{enumerate}`'
- '`\end{enumerate}`'
- '`\begin{description}`'
- '`\end{description}`'

Everything between '`%%PreScript:skip`' and '`%%PreScript:piks`' will be ignored in `PreScript`, so that there can be inserted command definitions for LaTeX exclusively.

8.3.2 Commands available in PreScript but not in LaTeX

The commands '`\textbi`' (for bold-italic) and '`\textsy`' (for symbol) do not exist in LaTeX. They should be defined in the preamble:

```
%%PreScript:skip
\newcommand{\textbi}[1]{\textbf{\textit{#1}}}
\newcommand{\textsy}[1]{#1}
%%PreScript:piks
```

There is no way in `PreScript` to get an automatic numbering. There is no equivalent to the LaTeX environment `enumerate`. But every command beginning by `\text` is doubled by a command beginning by '`\magic`'. `a2ps` behaves the same way on both families of commands. Hence, if one specifies that arguments of those functions should be ignored in the preamble of the LaTeX document, the numbering is emulated. For instance

```
\begin{enumerate}
\magicbf{1.}\item First line
\magicbf{2.}\item Second line
```



```
\end{enumerate}
```

will be treated the same way both in `PreScript` and LaTeX.

'`header`' and '`\footer`', are not understood by LaTeX.

Some symbols are available in `PreScript` but not in LaTeX:

'`\Alpha`', '`\apple`', '`\Beta`', '`\carriagereturn`', '`\Chi`', '`\Epsilon`', '`\Eta`', '`\florin`', '`\Iota`', '`\Kappa`', '`\Mu`', '`\Nu`', '`\Omicron`', '`\omicron`', '`\radicalex`', '`\register`', '`\Rho`', '`\suchthat`', '`\Tau`', '`\therefore`', '`\trademark`', '`\varUpsilon`', '`\Zeta`'.

A good example of a LaTeX preamble to get LaTeX to compile `PreScript` files is given by the report of `a2ps`.

8.4 Examples

`PreScript` and `a2ps` can be used for one-the-fly formating. For instance, on the '`passwd`' file:

```
ypcat passwd |
 awk -F: \
   '{print "\textbf{" $5 "} (" $1 ") \rightarrow\textit{" $7 "}"}'\
 | a2ps -Epre -P
```

9 Programming with the Library

a2ps offers to the programmer an access to its generating routines. This section documents the API.

But since this section is empty, or almost, if I were you, I would go in contrib/sample to see how it works...

9.1 Initialization and Closing of Calls to liba2ps

struct a2ps_job * **a2ps_job_new** (void) Function
Build a new storage unit for the library.

void **a2ps_read_sys_config** (struct a2ps_job * *job*) Function
Set *job* with the default settings defined in the configuration file of the system.

void **a2ps_read_config** (struct print_job * *job*, char * *path*, char * *filename*) Function
Set *job* with the default settings defined in the configuration file '*path*/*filename*'. *path* can be NULL, *filename* cannot.

9.2 Print Jobs

A print job should be seen as associated to a single output.

void **a2ps_open_output_session** (struct a2ps_job * *job*) Function
Initialize *job* for a new print job.

void **a2ps_close_output_session** (struct print_job * *job*) Function
Closes the current print job, and sends the output.

9.3 File Jobs

A file job should be seen as a single input.

void **a2ps_new_input_session** (struct a2ps_job * *job*, char * *name*) Function
Create and open a new file job in *job*. *name* can be NULL, in which case its name is defaulted to that of stdin.

void **a2ps_close_input_session** (struct print_job * *job*) Function
End the current input session.

9.4 Printing Functions

These are the functions to be used to send content to liba2ps.

void **a2ps_print_char** (struct print_job * *job*, unsigned char *c*, face_t *face*) Function
Print the char *c* in *face* in *job*.

void **a2ps_print_string** (struct print_job * *job*, unsigned char * *string*, face_t *face*) Function
Print the C string *string* in *face* in *job*.

void **a2ps_print_buffer** (struct print_job * *job*, unsigned char * *buffer*, size_t len, face_t *face*) Function
Print the *len* first characters contained in *buffer* with *face* into *job*.

10 Various

We collected in this chapter the things that had no place in the rest of the document.

10.1 Security issues

Note. I am not really aware of security problems. I am just making assumptions upon my poor knowledge, so if somebody sees things that should be reported here, or problem I'm not aware of, please contact us so that this note gets extended or fixed.

One should understand that any program that has not been written to be secure is never secure. It is of course the case of `a2ps`.

Do not panic, there are no reason for you to worry. Nevertheless we can yet imagine ways to obtain illegal rights thanks to some features of `a2ps`, especially virtual printers.

If `a2ps` is run by root, then the files it may create are owned by root. This can for instance happen if you install `a2ps` as one of `lp` or `lpr` filter.

Then, if one of the virtual printer creates say a shell script, it is owned by root too. With just a bit of habit, it should not be difficult then to access privileged access to the system.

In what conditions could it happen? **Only** if there are some printers defined in the system configuration file of `a2ps`, or in root's home directory '`.a2ps/a2psrc`'. Hence, make sure to carefully write the commands to the preconfigured printers.

As you can see, this is quite science fiction, nevertheless, you might wanted to know.

10.2 Non PostScript printers

Here are some tips we have on how to use a non PostScript printer. If somebody feels like writing a more precise documentation, he really is welcome.

10.2.1 From Clayton Weaver

(for a bj200ex on os2 from linux) This is a shell script called as a filter from an ":if=/usr/local/bin/filtername" entry in /etc/printcap on linux:

```
#!/bin/bash

gs -q -sDevice=bj200 -sOutputfile=- -r360 -dNOPAUSE -dSAFER -
```

The device is a gs device name, input is the ending "-" (stdin), output is stdout, -q is "no messages". Can be used with

```
lpr -Pprintername -Ffilter_path_and_name infile.ps
```

or just '`lpr -Pprintcap_printername infile.ps`' if the filter is specified in the printcap '`:if=`' entry for that "printcap_printername". Either way, lpr on linux runs it in a sub shell, so you don't want "exec" before the gs command (that kills the sub shell and leaves gs running with no way to communicate its stdout back to lpr).

You can print directly from gs, I've done it on OS/2 by specifying "lpt1" as the output device, but printing to a remote printer I need lpr and lpd to handle transmitting the job, so this way works better for me.

Regards, Clayton Weaver (`cgweav@eskimo.com`) (Seattle)

10.2.2 From Pierre Juillot

To print toto.ps on my desk-jet, I hit:

```
> gsdj toto.ps
```

where '`gsdj`' is

```
#!/bin/sh
#                              05-06-1996
# to print a PostScript file using GhostScript on the
# desk-jet 500 (dj) or the printer specified via  the
# environment variable LPOPTS
# using GhostScript
# exec gs -q -dNOPAUSE -sPAPERSIZE=a4 -sDEVICE=djet500 \
exec gs -q -dNOPAUSE -sPAPERSIZE=a4 -sDEVICE=deskjet \
     -sOutputFile="|lp -onb -ddj -or $LPOPTS " $* quit.ps
```

10.2.3 These tips in a2ps

To use these examples, first see Section 4.1.4 [Your Printers], page 29. The translations would be in one of your configuration files, say '`$HOME/.a2ps/a2psrc`':

```
Printer: lpt1 | gs -q -sDevice=bj200 -sOutputfile=- -r360 -dNOPAUSE \
        -dSAFER > lpt1
UnknownPrinter:  | gs -q -dNOPAUSE -sPAPERSIZE=a4 -sDEVICE=deskjet \
     -sOutputFile="|lp -onb -ddj -or #o" quit.ps
```

10.3 Frequently asked questions

Please, before sending us mail, make sure the problem you have is not known, and explained. Moreover, avoid using the mailing list for asking question about the options, etc. It has been built for announces and suggestions, not to contact the authors.

10.3.1 Relocation Error

When I have just compiled `a2ps`, and I first try it, I have weird errors such as:

```
gargantua ~T $ a2ps --help
ld.so.1: a2ps: fatal: relocation error: symbol not found:
       ps_comment_hook: referenced in a2ps
zsh: 7294 killed     a2ps --help
```

This is due to a mix between several versions of the `liba2ps`. To fix the problem, make sure to run '`make install`' in the '`lib/`' directory and then in the '`src/`' directory. Removing the previous '`liba2ps.so`' files may help.

10.3.2 Printer Errors

There are two ways that printing can fail: silently, or with an diagnostic.

First check that you did not give exotic options to an old printer (typically, avoid printing on two sides on a printer that does not support it). Avoid use of options '`-D`' (see Section 7.1 [Page Device Options], page 59) and '`-S`' (see Section 7.2 [Statusdict Options], page 59).

If the trouble persists, please try again but with the option '--debug' (a PostScript error handler is down-loaded), and then send us:

1. the file that gives problems
2. the error message that was printed.

10.3.3 Cannot Print in Duplex

If your printer is too old, then a2ps will not be able to send it the code it needs when '-s2' is specified. This is because your printer uses an old and not standardized interface for special features.

So you need to

1. specify that you want Duplex mode: '-s2',
2. remove at hand the standardized call to the Duplex feature: '-DDuplex',
3. add the non standard call to the Duplex features. Try '-Ssetduplexmode:true'.

Since this is painful to hit, a User Option (see Section 4.1.5 [Your Shortcuts], page 29) should help.

10.3.4 Printing goes beyond the frame of the paper

You are currently printing with a bad medium, for instance using A4 paper within a2ps, while your printer uses Letter paper. See Section 3.1.1 [Global behavior], page 11, option '--medium' for more.

10.3.5 How can I leave room to bind?

The option '--margin[=*size*]' is meant for this. See Section 3.1.2 [Sheets Options], page 13.

10.3.6 Why not having used yacc and such

There are several reasons why we decided not to use grammars to parse the files.

First it would have made the design of the style sheets much more tricky, and today a2ps would know only 4 or 5 languages.

Second, it limits the number of persons who could build a style sheet.

Third, mixing several parsers within one program is not easy. Moreover, its would have been ten times bigger.

Fourth, we did not feel the need for such a powerful tool: handling the keywords and the sequences is just what the users expect.

Fifth, any extension of a2ps would have required to recompile.

And last but not least, using a parser requires that the sources are syntactic bug free, what is too strong a requirement.

Nevertheless, PreScript gives the possibility to have on the one hand a syntactic parser which would produce PreScript code, and on the other hand, a2ps, which would make it PostScript. This schema seems to us a good compromise. If it is still not enough for you, you can use the library (see Chapter 9 [Programming with the Library], page 67).

10.3.7 Man pages

There are no man pages because now the documentation is written in Texinfo. What you are reading comes from this initial Texinfo file.

Nevertheless, man pages may be back from Hell the day a texi2roff filter appears.

10.3.8 What I get on the printer is long and incomprehensible

You are probably printing a PostScript file or equivalent. Try to print with '-Z': a2ps will try to do his best to find what is the program that can help you (see Section 4.1.9 [Your Delegations], page 31). In case of doubt, don't hesitate to save into a file, and check the content with GhostView, or such:

```
$ a2ps my_weird_file -Z -o mwf.ps
$ ghostview mwf.ps
```

If it is not correct, ask for help around you.

10.3.9 Printing stdin

a2ps prints the standard input if you give no file name, or if you gave '-' as file name. It cannot automatically pretty print, nor call delegations if you don't specify the language (see Section 4.4 [Style Sheet Files], page 34), or if you don't supply a name to the standard input ('--stdin=*name*') with which it could guess the language.

10.3.10 Can I change the base font

Well, yes. See Section 7.4 [Designing PostScript prologues], page 60, for details. Make sure that all the information a2ps needs is available (see Section 4.3 [Font Files], page 33).

10.4 What next?

There are some features we would like to implement. Some are the following ones. If you feel like helping, don't hesitate to contact us.

10.4.1 EPSF

a2ps should be able to generate EPS files in a short while.

10.4.2 Spelling checking, syntax fixing.

When printing a human file (only English may be supported), spelling should be checked. When pretty-printing, syntax errors should be underlined.

No just kidding :). Do it yourself...

Appendix A Genesis

Here are some words on a2ps and its history.

A.1 History

The initial version was a shell program written by Evan Kirshenbaum (evan@csli). It was very slow and contained many bugs.

A new version was written in C by Miguel Santana (Miguel.Santana@st.com) to improve execution speed and portability. Many new features and improvements have been added since this first version. Many contributions (changes, fixes, ideas) were done by a2ps users in order to improve it.

From the latest version from Miguel Santana (4.3), Emmanuel Briot implemented bold faces for keywords in Ada, C and C++.

From that version, Akim Demaille (demaille@inf.enst.fr) generalized the pretty-printing capabilities, implemented more languages support, and other features.

A.2 Contributors

Since the very beginning of a2ps, many people have brought help:

- Oscar Nierstrasz (oscar@iam.unibe.ch)
- Tim Clark (T.Clark@warwick.ac.uk)
- tullemans@apolloway.prl.philips.nl
- Johan Vromans (jv@mh.nl)
- craig.r.stevenson@att.com
- erikt@cs.umu.se
- wstahw@lso.win.tue.nl
- mai@wolfen.cc.uow.oz
- Johan Garpendahl (garp@isy.liu.se)
- John Interrante (interran@uluru.stanford.edu)
- Chris Adamo (adamo@ll.mit.edu)
- Larry Barbieri (lbarbieri@ll.mit.edu)
- Denis Girou (Denis.Girou@idris.fr)
- Thomas Parmelan (Thomas.Parmelan@efrei.fr)
- Paul Shum (pshum@ali.bc.ca)
- Graham Jenkins (grahamj@webnet.com.au), thanks to whom a2ps (but also GNU Libtool) is ported onto Pyramid Systems.
- Didier Verna (verna@inf.enst.fr) wrote a2ps-compile-regexp for the a2ps-mode.
- Jim Diamond (zsd@axe.nstn.ca).
- Christian Mondrup (scancm@biobase.dk) who regularly tracks down bugs appearing on DEC architectures.

- Michael Taeschner (Michael.Taeschner@dlr.de) who tracked the same bug, but with another look, on Irix 6.2.
- Peter Bartke (bartke@inf.fu-berlin.de)

A.3 Translators

Some people worked on the translation of a2ps:

- Daniele Ghiotti (Italian)
- Tomek Burdziak (Polish)
- Miguel A. Varo (mvaro@dlsi.ua.es) (Maintainer for Spanish and Catalan)
- Michael Wiedmann (mw@miwie.in-berlin.de) (Maintainer for German)
- Christian Kirsch (ck@held.mind.de) (German)
- Erwin Dieterich (debaycrk@ibmmail.com) (German)
- Juliusz Chroboczek (jec@dcs.ed.ac.uk) (Polish). He is also the author of Ogonkify (see section "Overview" in *Ogonkify manual*).
- Marcel van der Laan (marcel@spase.nl) (Dutch)
- Lorenzo M. Catucci (lorenzo@argon.roma2.infn.it) (Italian)
- Choi Jun Ho (junker@jazz.snu.ac.kr) (Korean)
- Turgut Uyar (uyar@cs.itu.edu.tr) (Turkish)

A.4 Thanks

Patrick Andries, from Alis Technologies inc.[1] , authorized us to include pieces of very nice texts about the encodings. We strongly recommend that you go and read these pages: there is a lot to learn.

Juliusz Chroboczek worked a lot on the integration of the products of Ogonkify (such as Latin 2 etc. fonts) in a2ps. Without his help, and the time is devoted to both a2ps and ogonkify, many non west-European people would still be unable to print easily texts written in their mother tongue.

Denis Girou brought a constant and valuable support through out the genesis of pretty-printing a2ps. His comments on both the program and the documentation are the origin of many pleasant features (such as '--prologue').

We would like to thank Markku Rossi (the author of GNU enscript), for the advises he gave, and for portions of his code which are now used in a2ps. Even portions of this very documentation are taken from the GNU enscript man pages.

[1] http://www.alis.com/

Appendix B Copying

The subroutines and source code in the a2ps package are "free"; this means that everyone is free to use them and free to redistribute them on a free basis. The a2ps-related programs are not in the public domain; they are copyrighted and there are restrictions on their distribution, but these restrictions are designed to permit everything that a good cooperating citizen would want to do. What is not allowed is to try to prevent others from further sharing any version of these programs that they might get from you.

Specifically, we want to make sure that you have the right to give away copies of the programs that relate to a2ps, that you receive source code or else can get it if you want it, that you can change these programs or use pieces of them in new free programs, and that you know you can do these things.

To make sure that everyone has such rights, we have to forbid you to deprive anyone else of these rights. For example, if you distribute copies of the a2ps-related code, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must tell them their rights.

Also, for our own protection, we must make certain that everyone finds out that there is no warranty for the programs that relate to a2ps. If these programs are modified by someone else and passed on, we want their recipients to know that what they have is not what we distributed, so that any problems introduced by others will not reflect on our reputation.

The precise conditions of the licenses for the programs currently being distributed that relate to a2ps are found in the General Public Licenses that accompany them.

Concept Index

Table of Contents